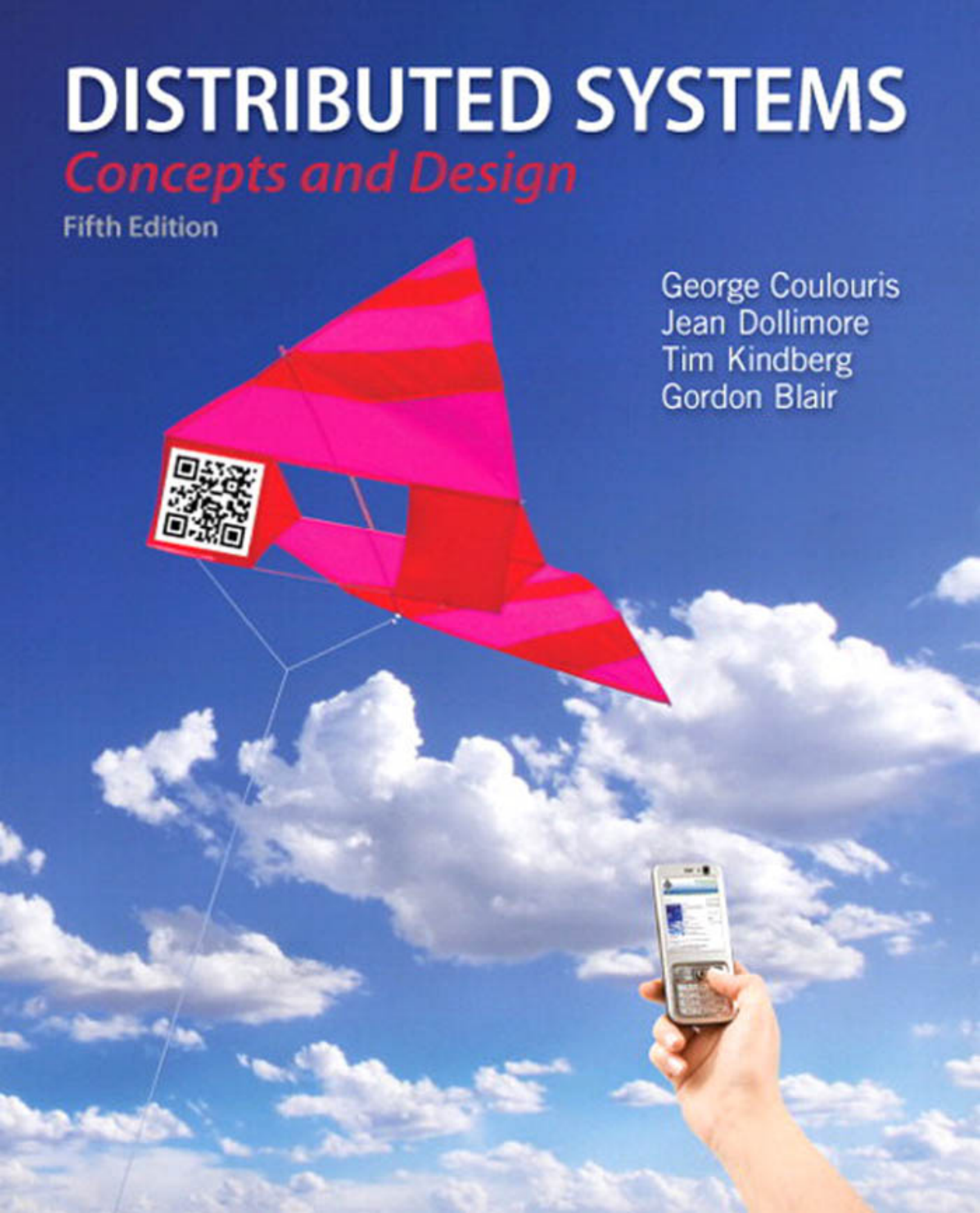
DISTRIBUTED SYSTEMS

Concepts and Design

Fifth Edition

George Coulouris
Jean Dollimore
Tim Kindberg
Gordon Blair



6

INDIRECT COMMUNICATION

This chapter completes our tour of communication paradigms by examining indirect communication; it builds on our studies of interprocess communication and remote invocation in Chapters 4 and 5, respectively. The essence of indirect communication is to communicate through an intermediary and hence have no direct coupling between the sender and the one or more receivers. The important concepts of space and time uncoupling are also introduced.

The chapter examines a range of indirect communication techniques:

- group communication, in which communication is via a group abstraction with the sender unaware of the identity of the recipients;
- publish-subscribe systems, a family of approaches that all share the common characteristic of disseminating events to multiple recipients through an intermediary;
- message queue systems, wherein messages are directed to the familiar abstraction of a queue with receivers extracting messages from such queues;
- shared memory–based approaches, including distributed shared memory and tuple space approaches, which present an abstraction of a global shared memory to programmers.

Case studies are used throughout the chapter to illustrate the main concepts introduced.

6.1 Introduction

This chapter concludes our examination of communication paradigms by examining indirect communication, building on the studies of interprocess communication and remote invocation in Chapters 4 and 5, respectively. Indirection is a fundamental concept in computer science, and its ubiquity and importance are captured nicely by the following quote, which emerged from the Titan Project at the University of Cambridge and is attributable to Roger Needham, Maurice Wilkes and David Wheeler:

> All problems in computer science can be solved by another level of indirection.

In terms of distributed systems, the concept of indirection is increasingly applied to communication paradigms.

Indirect communication is defined as communication between entities in a distributed system through an intermediary with no direct coupling between the sender and the receiver(s). The precise nature of the intermediary varies from approach to approach, as will be seen in the rest of this chapter. In addition, the precise nature of coupling varies significantly between systems, and again this will be brought out in the text that follows. Note the optional plural associated with the receiver; this signifies that many indirect communication paradigms explicitly support one-to-many communication.

The techniques considered in Chapters 4 and 5 are all based on a direct coupling between a sender and a receiver, and this leads to a certain amount of rigidity in the system in terms of dealing with change. To illustrate this, consider a simple client-server interaction. Because of the direct coupling, it is more difficult to replace a server with an alternative one offering equivalent functionality. Similarly, if the server fails, this directly affects the client, which must explicitly deal with the failure. In contrast, indirect communication avoids this direct coupling and hence inherits interesting properties. The literature refers to two key properties stemming from the use of an intermediary:

Space uncoupling, in which the sender does not know or need to know the identity of the receiver(s), and vice versa. Because of this space uncoupling, the system developer has many degrees of freedom in dealing with change: participants (senders or receivers) can be replaced, updated, replicated or migrated.

Time uncoupling, in which the sender and receiver(s) can have independent lifetimes. In other words, the sender and receiver(s) do not need to exist at the same time to communicate. This has important benefits, for example, in more volatile environments where senders and receivers may come and go.

For these reasons, indirect communication is often used in distributed systems where change is anticipated – for example, in mobile environments where users may rapidly connect to and disconnect from the global network – and must be managed to provide more dependable services. Indirect communication is also heavily used for event dissemination in distributed systems where the receivers may be unknown and liable to change – for example, in managing event feeds in financial systems, as featured in Chapter 1. Indirect communication is also exploited in key parts of the Google infrastructure, as discussed in the major case study in Chapter 21.

Figure 6.1 Space and time coupling in distributed systems

	Time-coupled	*Time-uncoupled*
Space coupling	*Properties*: Communication directed towards a given receiver or receivers; receiver(s) must exist at that moment in time *Examples*: Message passing, remote invocation (see Chapters 4 and 5)	*Properties*: Communication directed towards a given receiver or receivers; sender(s) and receiver(s) can have independent lifetimes *Examples*: See Exercise 6.3
Space uncoupling	*Properties*: Sender does not need to know the identity of the receiver(s); receiver(s) must exist at that moment in time *Examples*: IP multicast (see Chapter 4)	*Properties*: Sender does not need to know the identity of the receiver(s); sender(s) and receiver(s) can have independent lifetimes *Examples*: Most indirect communication paradigms covered in this chapter

The discussion above charts the advantages associated with indirect communication. The main disadvantage is that there will inevitably be a performance overhead introduced by the added level of indirection. Indeed, the quote above on indirection is often paired by the following quote, attributable to Jim Gray:

> There is no performance problem that cannot be solved by eliminating a level of indirection.

In addition, systems developed using indirect communication can be more difficult to manage precisely because of the lack of any direct (space or time) coupling.

A closer look at space and time uncoupling • It may be assumed that indirection implies *both* space and time uncoupling, but this is not always the case. The precise relationship is summarized in Figure 6.1.

From this table, it is clear that most of the techniques considered in this book are either coupled in both time and space or indeed uncoupled in both dimensions. The top-left box represents the communication paradigms featured in Chapters 4 and 5 where communication is direct with no space or time uncoupling. For example, message passing is both directed towards a particular entity and requires the receiver to be present at the time of the message send (but see Exercise 6.2 for an added dimension introduced by DNS name resolution). The range of remote invocation paradigms are also coupled in both space and time. The bottom-right box represents the main indirect communication paradigms that exhibit both properties. A small number of communication paradigms sit outside these two areas:

- IP multicast, as featured in Chapter 4, is space-uncoupled but time-coupled. It is space-uncoupled because messages are directed towards the multicast group, not any particular receiver. It is time-coupled, though, as all receivers must exist at the time of the message send to receive the multicast. Some implementations of group communication and indeed publish-subscribe systems, also fall into this category (see Section 6.6). This example illustrates the importance of *persistency* in the

communication channel to achieve time uncoupling – that is, the communication paradigm must store messages so that they can be delivered when the receiver(s) is ready to receive. IP multicast does not support this level of persistency.

- The case in which communication is space-coupled but time-uncoupled is more subtle. Space coupling implies that the sender knows the identity of a specific receiver or receivers, but time uncoupling implies that the receiver or receivers need not exist at the time of sending. Exercises 6.3 and 6.4 invite the reader to consider whether such a paradigm exists or could be constructed.

Returning to our definition, we treat *all* paradigms that involve an intermediary as indirect and recognize that the precise level of coupling will vary from system to system. We revisit the properties of different indirect communication paradigms in Section 6.6, once we have had a chance to study the precise characteristics of each approach.

The relationship with asynchronous communication • Note that, to fully understand this area, it is important to distinguish between asynchronous communication (as defined in Chapter 4) and time uncoupling. In asynchronous communication, a sender sends a message and then continues (without blocking), and hence there is no need to meet in time with the receiver to communicate. Time uncoupling adds the extra dimension that the sender and receiver(s) can have independent existences; for example, the receiver may not exist at the time communication is initiated. Eugster *et al.* also recognize the important distinction between asynchronous communication (synchronization uncoupling) and time uncoupling [2003].

Many of the techniques examined in this chapter are time-uncoupled and asynchronous, but a few, such as the *MessageDispatcher* and *RpcDispatcher* operations in JGroups, discussed in Section 6.2.3, offer a synchronous service over indirect communication.

The rest of the chapter examines specific examples of indirect communication starting with group communication in Section 6.2. Section 6.3 then examines the fundamentals of publish-subscribe systems, with Section 6.4 examining the contrasting approach offered by message queues. Following this, Section 6.5 considers approaches based on shared memory abstractions, specifically distributed shared memory and tuple space–based approaches.

6.2 Group communication

Group communication provides our first example of an indirect communication paradigm. *Group communication* offers a service whereby a message is sent to a group and then this message is delivered to all members of the group. In this action, the sender is not aware of the identities of the receivers. Group communication represents an abstraction over multicast communication and may be implemented over IP multicast or an equivalent overlay network, adding significant extra value in terms of managing group membership, detecting failures and providing reliability and ordering guarantees. With the added guarantees, group communication is to IP multicast what TCP is to the point-to-point service in IP.

Group communication is an important building block for distributed systems, and particularly reliable distributed systems, with key areas of application including:

- the reliable dissemination of information to potentially large numbers of clients, including in the financial industry, where institutions require accurate and up-to-date access to a wide variety of information sources;
- support for collaborative applications, where again events must be disseminated to multiple users to preserve a common user view – for example, in multiuser games, as discussed in Chapter 1;
- support for a range of fault-tolerance strategies, including the consistent update of replicated data (as discussed in detail in Chapter 18) or the implementation of highly available (replicated) servers;
- support for system monitoring and management, including for example load balancing strategies.

We look at group communication in more detail below, examining the programming model offered and the associated implementation issues. We examine the JGroups toolkit as a case study of a group communication service.

6.2.1 The programming model

In group communication, the central concept is that of a *group* with associated *group membership*, whereby processes may *join* or *leave* the group. Processes can then send a message to this group and have it propagated to all members of the group with certain guarantees in terms of reliability and ordering. Thus, group communication implements *multicast* communication, in which a message is sent to all the members of the group by a single operation. Communication to *all* processes in the system, as opposed to a subgroup of them, is known as *broadcast*, whereas communication to a single process is known as *unicast*.

The essential feature of group communication is that a process issues only one multicast operation to send a message to each of a group of processes (in Java this operation is *aGroup.send(aMessage)*) instead of issuing multiple send operations to individual processes.

The use of a single multicast operation instead of multiple send operations amounts to much more than a convenience for the programmer: it enables the implementation to be efficient in its utilization of bandwidth. It can take steps to send the message no more than once over any communication link, by sending it over a distribution tree; and it can use network hardware support for multicast where this is available. The implementation can also minimize the total time taken to deliver the message to all destinations, as compared with transmitting it separately and serially.

To see these advantages, compare the bandwidth utilization and the total transmission time taken when sending the same message from a computer in London to two computers on the same Ethernet in Palo Alto, (a) by two separate UDP sends, and (b) by a single IP multicast operation. In the former case, two copies of the message are sent independently, and the second is delayed by the first. In the latter case, a set of multicast-aware routers forward a single copy of the message from London to a router on the destination LAN in California. That router then uses hardware multicast (provided by the Ethernet) to deliver the message to both destinations at once, instead of sending it twice.

The use of a single multicast operation is also important in terms of delivery guarantees. If a process issues multiple independent send operations to individual processes, then there is no way for the implementation to provide guarantees that affect the group of processes as a whole. If the sender fails halfway through sending, then some members of the group may receive the message while others do not. In addition, the relative ordering of two messages delivered to any two group members is undefined. Group communication, however, has the potential to offer a range of guarantees in terms of reliability and ordering, as discussed in Section 6.2.2 below.

Group communication has been the subject of many research projects, including the V-system [Cheriton and Zwaenepoel 1985], Chorus [Rozier *et al.* 1988], Amoeba [Kaashoek *et al.* 1989; Kaashoek and Tanenbaum 1991], Trans/Total [Melliar-Smith *et al.* 1990], Delta-4 [Powell 1991], Isis [Birman 1993], Horus [van Renesse *et al.* 1996], Totem [Moser *et al.* 1996] and Transis [Dolev and Malki 1996] – and we shall cite other notable work in the course of this chapter and indeed throughout the book (particularly in Chapters 15 and 18).

Process groups and object groups • Most work on group services focuses on the concept of *process groups*, that is, groups where the communicating entities are processes. Such services are relatively low-level in that:

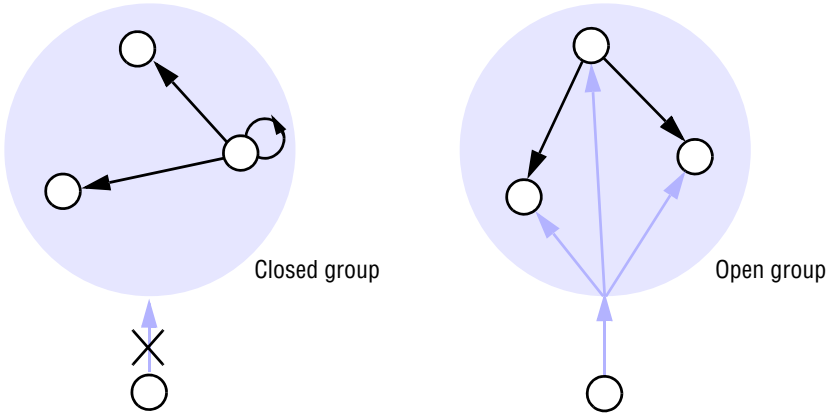
- Messages are delivered to processes and no further support for dispatching is provided.
- Messages are typically unstructured byte arrays with no support for marshalling of complex data types (as provided, for example, in RPC or RMI – see Chapter 5).

The level of service provided by process groups is therefore similar to that of sockets, as discussed in Chapter 4. In contrast, *object groups* provide a higher-level approach to group computing. An object group is a collection of objects (normally instances of the same class) that process the same set of invocations concurrently, with each returning responses. Client objects need not be aware of the replication. They invoke operations on a single, local object, which acts as a proxy for the group. The proxy uses a group communication system to send the invocations to the members of the object group. Object parameters and results are marshalled as in RMI and the associated calls are dispatched automatically to the right destination objects/methods.

Electra [Maffeis 1995] is a CORBA-compliant system that supports object groups. An Electra group can be interfaced to any CORBA-compliant application. Electra was originally built on top of the Horus group communication system, which it uses to manage the membership of the group and to multicast invocations. In 'transparent mode', the local proxy returns the first available response to a client object. In 'non-transparent mode', the client object can access all the responses returned by the group members. Electra uses an extension of the standard CORBA Object Request Broker interface, with functions for creating and destroying object groups and managing their membership. Eternal [Moser *et al.* 1998] and the Object Group Service [Guerraoui *et al.* 1998] also provide CORBA-compliant support for object groups.

Despite the promise of object groups, however, process groups still dominate in terms of usage. For example, the popular JGroups toolkit, discussed in Section 6.2.3, is a classical process group approach.

Figure 6.2 Open and closed groups



Other key distinctions • A wide range of group communication services has been developed, and they vary in the assumptions they make:

Closed and open groups: A group is said to be *closed* if only members of the group may multicast to it (Figure 6.2). A process in a closed group delivers to itself any message that it multicasts to the group. A group is *open* if processes outside the group may send to it. (The categories 'open' and 'closed' also apply with analogous meanings to mailing lists.) Closed groups of processes are useful, for example, for cooperating servers to send messages to one another that only they should receive. Open groups are useful, for example, for delivering events to groups of interested processes.

Overlapping and non-overlapping groups: In *overlapping groups*, entities (processes or objects) may be members of multiple groups, and non-overlapping groups imply that membership does not overlap (that is, any process belongs to at most one group). Note that in real-life systems, it is realistic to expect that group membership will overlap.

Synchronous and asynchronous systems: There is a requirement to consider group communication in both environments.

Such distinctions can have a significant impact on the underlying multicast algorithms. For example, some algorithms assume that groups are closed. The same effect as openness can be achieved with a closed group by picking a member of the group and sending it a message (one-to-one) for it to multicast to its group. Rodrigues *et al.* [1998] discuss multicast to open groups. Issues related to open and closed groups arise in Chapter 15, when algorithms for reliability and ordering are considered. That chapter also considers the impact of overlapping groups and whether the system is synchronous or asynchronous on such protocols.

6.2.2 Implementation issues

We now turn to implementation issues for group communication services, discussing the properties of the underlying multicast service in terms of reliability and ordering and also the key role of group membership management in dynamic environments, where processes can join and leave or fail at any time.

Reliability and ordering in multicast • In group communication, all members of a group must receive copies of the messages sent to the group, generally with delivery guarantees. The guarantees include agreement on the set of messages that every process in the group should receive and on the delivery ordering across the group members.

Group communication systems are extremely sophisticated. Even IP multicast, which provides minimal delivery guarantees, requires a major engineering effort.

So far, we have discussed reliability and ordering in rather general terms. We now look in more detail at what such properties mean.

Reliability in one-to-one communication was defined in Section 2.4.2 in terms of two properties: integrity (the message received is the same as the one sent, and no messages are delivered twice) and validity (any outgoing message is eventually delivered). The interpretation for *reliable multicast* builds on these properties, with *integrity* defined the same way in terms of delivering the message correctly at most once, and *validity* guaranteeing that a message sent will eventually be delivered. To extend the semantics to cover delivery to multiple receivers, a third property is added – that of *agreement*, stating that if the message is delivered to one process, then it is delivered to all processes in the group.

As well as reliability guarantees, group communication demands extra guarantees in terms of the relative ordering of messages delivered to multiple destinations. Ordering is not guaranteed by underlying interprocess communication primitives. For example, if multicast is implemented by a series of one-to-one messages, they may be subject to arbitrary delays. Similar problems may occur if using IP multicast. To counter this, group communication services offer *ordered multicast*, with the option of one or more of the following properties (with hybrid solutions also possible):

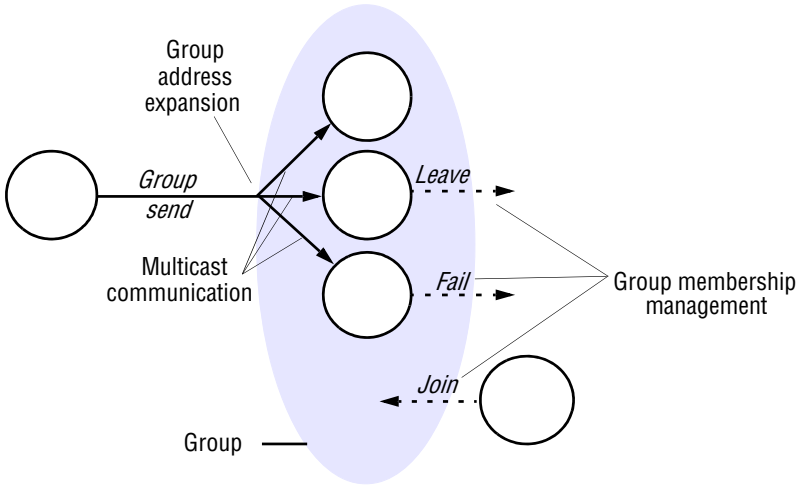
FIFO ordering: First-in-first-out (FIFO) ordering (also referred to as source ordering) is concerned with preserving the order from the perspective of a sender process, in that if a process sends one message before another, it will be delivered in this order at all processes in the group.

Causal ordering: Causal ordering takes into account causal relationships between messages, in that if a message *happens before* another message in the distributed system this so-called causal relationship will be preserved in the delivery of the associated messages at all processes (see Chapter 14 for a detailed discussion of the meaning of 'happens before').

Total ordering: In total ordering, if a message is delivered before another message at one process, then the same order will be preserved at all processes.

Reliability and ordering are examples of coordination and agreement in distributed systems, and hence further consideration of this is deferred to Chapter 15, which focuses exclusively on this topic. In particular, Chapter 15 provides more complete definitions

Figure 6.3 The role of group membership management



of integrity, validity, agreement and the various ordering properties and also examines in detail algorithms to realize reliable and ordered multicast.

Group membership management • The key elements of group communication management are summarized in Figure 6.3, which shows an open group. This diagram illustrates the important role of group membership management in maintaining an accurate *view* of the current membership, given that entities may join, leave or indeed fail. In more detail, a group membership service has four main tasks:

Providing an interface for group membership changes: The membership service provides operations to create and destroy process groups and to add or withdraw a process to or from a group. In most systems, a single process may belong to several groups at the same time (overlapping groups, as defined above). This is true of IP multicast, for example.

Failure detection: The service monitors the group members not only in case they should crash, but also in case they should become unreachable because of a communication failure. The detector marks processes as *Suspected* or *Unsuspected*. The service uses the failure detector to reach a decision about the group's membership: it excludes a process from membership if it is suspected to have failed or to have become unreachable.

Notifying members of group membership changes: The service notifies the group's members when a process is added, or when a process is excluded (through failure or when the process is deliberately withdrawn from the group).

Performing group address expansion: When a process multicasts a message, it supplies the group identifier rather than a list of processes in the group. The membership management service expands the identifier into the current group

membership for delivery. The service can coordinate multicast delivery with membership changes by controlling address expansion. That is, it can decide consistently where to deliver any given message, even though the membership may be changing during delivery.

Note that IP multicast is a weak case of a group membership service, with some but not all of these properties. It does allow processes to join or leave groups dynamically and it performs address expansion, so that senders need only provide a single IP multicast address as the destination for a multicast message. But IP multicast does not itself provide group members with information about current membership, and multicast delivery is not coordinated with membership changes. Achieving these properties is complex and requires what is known as *view-synchronous group communication*. Further consideration of this important issue is deferred to Chapter 18, which discusses the maintenance of group views and how to realize view-synchronous group communication in the context of supporting replication in distributed systems.

In general, the need to maintain group membership has a significant impact on the utility of group-based approaches. In particular, group communication is most effective in small-scale and static systems and does not operate as well in larger-scale environments or environments with a high degree of volatility. This can be traced to the need for a form of synchrony assumption. Ganesh et al [2003] present a more probabilistic approach to group membership designed to operate in more large-scale and dynamic environments, using an underlying gossip protocol (see Section 10.5.3). Researchers have also developed group membership protocols specifically for ad hoc networks and mobile environments [Prakash and Baldoni 1998; Roman *et al*. 2001; Liu *et al*. 2005].

6.2.3 Case study: the JGroups toolkit

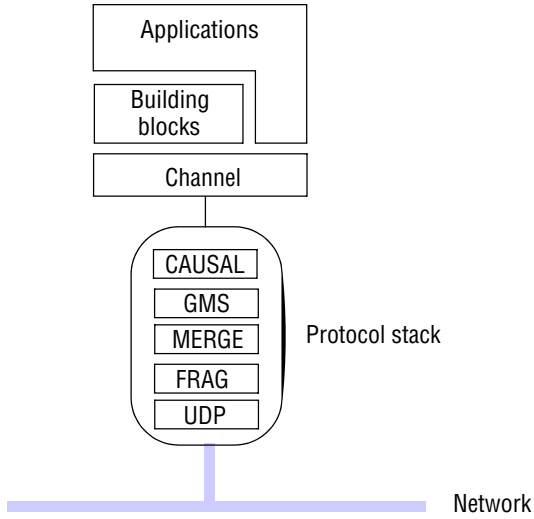
JGroups is a toolkit for reliable group communication written in Java. The toolkit is a part of the lineage of group communication tools that have emerged from Cornell University, building on the fundamental concepts developed in ISIS [Birman 1993], Horus [van Renesse *et al.* 1996] and Ensemble [van Renesse *et al.* 1998]. The toolkit is now maintained and developed by the JGroups open source community [www.jgroups.org], which is part of the JBoss middleware community, as discussed in Chapter 8 [www.jboss.org].

JGroups supports process groups in which processes are able to join or leave a group, send a message to all members of the group or indeed to a single member, and receive messages from the group. The toolkit supports a variety of reliability and ordering guarantees, which are discussed in more detail below, and also offers a group membership service.

The architecture of JGroups is shown in Figure 6.4, which shows the main components of the JGroups implementation:

- *Channels* represent the most primitive interface for application developers, offering the core functions of joining, leaving, sending and receiving.
- *Building blocks* offer higher-level abstractions, building on the underlying service offered by channels.

Figure 6.4 The architecture of JGroups



- The *protocol stack* provides the underlying communication protocol, constructed as a stack of composable protocol layers.

We look at each in turn below.

Channels • A process interacts with a group through a *channel* object, which acts as a handle onto a group. When created, it is disconnected, but a subsequent *connect* operation binds that handle to a particular named group; if the named group does not exist, it is implicitly created at the time of the first connect. To leave the group, the process executes the corresponding *disconnect* operation. A *close* operation is also provided to render the channel unusable. Note that a channel can only be connected to one group at a time; if a process wants to connect to two or more groups, it must create multiple channels. When connected, a process can send or receive via a channel. Messages are sent by reliable multicast, with the precise semantics defined by the protocol stack deployed (as discussed further below).

A range of other operations are defined on channels, most notably to return management information associated with the channel. For example, *getView* returns the current view defined in terms of the current member list, while *getState* returns the historical application state associated with the group (this can be used, for example, by a new group member to catch up with previous events).

Note that the term *channel* should not be confused with *channel-based publish-subscribe*, as introduced in Section 6.3.1. A channel in JGroups is synonymous with an instance of a group as defined in Section 6.2.1.

Figure 6.5 Java class *FireAlarmJG*

```
import org.jgroups.JChannel;

public class FireAlarmJG {
    public void raise() {
        try {
            JChannel channel = new JChannel();
            channel.connect("AlarmChannel");
            Message msg = new Message(null, null, "Fire!");
            channel.send(msg);
        }
        catch(Exception e) {
        }
    }
}
```

We illustrate the use of channels further by a simple example, a service whereby an intelligent fire alarm can send a "Fire!" multicast message to any registered receivers. The code for the fire alarm is as shown in Figure 6.5.

When an alarm is raised, the first step is to create a new instance of *JChannel* (the class representing channels in JGroups) and then connect to a group called *AlarmChannel*. If this is the first connect, then the group will be created at this stage (unlikely in this example, or the alarm is not going to be very effective). The constructor for a message takes three parameters: the destination, the source and the payload. In this case, the destination is *null,* which specifies that the message is to be sent to all members (if an address is specified, it is sent to that address only). The source is also *null*; this need not be provided in JGroups as it will be included automatically. The payload is an unstructured byte array that is delivered to all members of the group through the *send* method. The code to create a new instance of the *FireAlarmJG* class and then raise an alarm would be:

```
FireAlarmJG alarm = new FireAlarmJG();
alarm.raise();
```

The corresponding code for the receiver end has a similar structure and is shown in Figure 6.6. In this case, however, after connecting a *receive* method is called. This method takes one parameter, a timeout. If it is set to zero, as in this case, the receive message will block until a message is received. Note that in JGroups incoming messages are buffered and *receive* returns the top element in the buffer. If no messages are present, then *receive* blocks awaiting the next message. Strictly speaking, *receive* can return a range of object types – for example, notification of a change in membership or of a suspected failure of a group member (hence the cast to *Message* above).

A given receiver must include the following code to await an alarm:

```
FireAlarmConsumerJG alarmCall = new FireAlarmConsumerJG();
String msg = alarmCall.await();
System.out.println("Alarm received: " + msg);
```

Figure 6.6 Java class *FireAlarmConsumerJG*

```
import org.jgroups.JChannel;

public class FireAlarmConsumerJG {
    public String await() {
        try {
            JChannel channel = new JChannel();
            channel.connect("AlarmChannel");
            Message msg = (Message) channel.receive(0);
            return (String) msg.GetObject();
        }
        catch(Exception e) {
            return null;
        }
    }
}
```

Building blocks • Building blocks are higher-level abstractions on top of the channel class discussed above. Channels are similar in level to sockets. Building blocks are analogous to the more advanced communication paradigms discussed in Chapter 5, offering support for commonly occurring patterns of communication (but in this case targeted at multicast communication). Examples of building blocks in JGroups are:

- *MessageDispatcher* is the most intuitive of the building blocks offered in JGroups. In group communication, it is often useful for a sender to send a message to a group and then wait for some or all of the replies. *MessageDispatcher* supports this by providing a *castMessage* method that sends a message to a group and blocks until a specified number of replies are received (for example, until a specified number *n*, a majority, or all messages are received).
- *RpcDispatcher* takes a specific method (together with optional parameters and results) and then invokes this method on all objects associated with a group. As with *MessageDispatcher*, the caller can block awaiting some or all of the replies.
- *NotificationBus* is an implementation of a distributed event bus, in which an event is any serializable Java object. This class is often used to implement consistency in replicated caches.

The protocol stack • JGroups follows the architectures offered by Horus and Ensemble by constructing protocol stacks out of protocol layers (initially referred to as micro-protocols in the literature [van Renesse *et al.* 1996, 1998]). In this approach, a protocol is a bidirectional stack of protocol layers with each layer implementing the following two methods:

```
public Object up (Event evt);
public Object down (Event evt);
```

Protocol processing therefore happens by passing events up and down the stack. In JGroups, events may be incoming or outgoing messages or management events, for example related to view changes. Each layer can carry out arbitrary processing on the

message, including modifying its contents, adding a header or indeed dropping or re-ordering the message.

To illustrate the concept further, let us examine the protocol stack shown in Figure 6.4. This shows a protocol that consists of five layers:

- The layer referred to as UDP is the most common transport layer in JGroups. Note that, despite the name, this is not entirely equivalent to the UDP protocol; rather, the layer utilizes IP multicast for sending to all members in a group and UDP datagrams specifically for point-to-point communication. This layer therefore assumes that IP multicast is available. If it is not, the layer can be configured to send a series of unicast messages to members, relying on another layer for membership discovery (in particular, a layer known as PING). For larger-scale systems operating over wide area networks, a TCP layer may be preferred (using the TCP protocol to send unicast messages and again relying on PING for membership discovery).
- FRAG implements message packetization and is configurable in terms of the maximum message size (8,192 bytes by default).
- MERGE is a protocol that deals with unexpected network partitioning and the subsequent merging of subgroups after the partition. A series of alternative merge layers are actually available, ranging from the simple to ones that deal with, for example, state transfer.
- GMS implements a group membership protocol to maintain consistent views of membership across the group (see Chapter 18 for further details of algorithms for group membership management).
- CAUSAL implements causal ordering, introduced in Section 6.2.2 (and discussed further in Chapter 15).

A wide range of other protocol layers are available, including protocols for FIFO and total ordering, for membership discovery and failure detection, for encryption of messages and for implementing flow-control strategies (see the JGroups web site for details [www.jgroups.org]). Note that because all layers implement the same interface, they can be combined in any order, although many of the resultant protocol stacks would not make sense. All members of a group must share the same protocol stack.

6.3 Publish-subscribe systems

We now turn our attention to the area of *publish-subscribe systems* [Baldoni and Virgillito 2005], sometimes also referred to as *distributed event-based systems* [Muhl *et al*. 2006]. These are the most widely used of all the indirect communication techniques discussed in this chapter. Chapter 1 has already highlighted that many classes of system are fundamentally concerned with the communication and processing of events (for example financial trading systems). More specifically, whereas many systems naturally map onto a request-reply or a remote invocation pattern of interaction as discussed in Chapter 5, many do not and are more naturally modelled by the more decoupled and reactive style of programming offered by events.

A publish-subscribe system is a system where *publishers* publish structured events to an event service and *subscribers* express interest in particular events through *subscriptions* which can be arbitrary patterns over the structured events. For example, a subscriber could express an interest in all events related to this textbook, such as the availability of a new edition or updates to the related web site. The task of the publish-subscribe system is to match subscriptions against published events and ensure the correct delivery of *event notifications*. A given event will be delivered to potentially many subscribers, and hence publish-subscribe is fundamentally a one-to-many communications paradigm.

Applications of publish-subscribe systems • Publish-subscribe systems are used in a wide variety of application domains, particularly those related to the large-scale dissemination of events. Examples include:

- financial information systems;
- other areas with live feeds of real-time data (including RSS feeds);
- support for cooperative working, where a number of participants need to be informed of events of shared interest;
- support for ubiquitous computing, including the management of events emanating from the ubiquitous infrastructure (for example, location events);
- a broad set of monitoring applications, including network monitoring in the Internet.

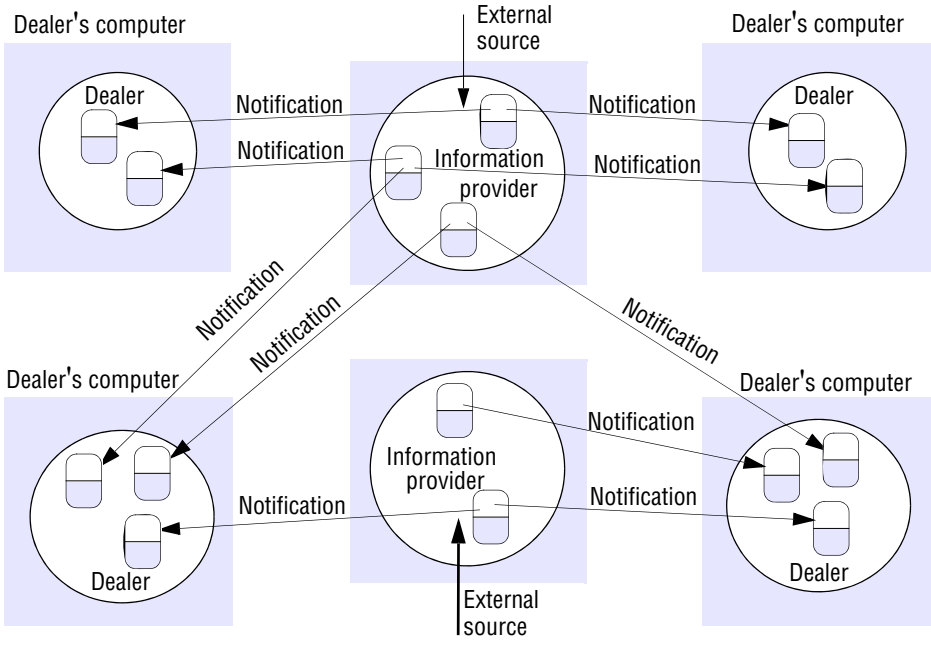
Publish-subscribe is also a key component of Google's infrastructure, including for example the dissemination of events related to advertisements, such as 'ad clicks', to interested parties (see Chapter 21).

To illustrate the concept further, we consider a simple dealing room system as an example of the broader class of financial information systems.

Dealing room system: Consider a simple dealing room system whose task is to allow dealers using computers to see the latest information about the market prices of the stocks they deal in. The market price for a single named stock is represented by an associated object. The information arrives in the dealing room from several different external sources in the form of updates to some or all of the objects representing the stocks and is collected by processes we call *information providers*. Dealers are typically interested only in their own specialist stocks. A dealing room system could be implemented by processes with two different tasks:

- An information provider process continuously receives new trading information from a single external source. Each of the updates is regarded as an event. The information provider publishes such events to the publish-subscribe system for delivery to all of the dealers who have expressed an interest in the corresponding stock. There will be a separate information provider process for each external source.
- A dealer process creates a subscription representing each named stock that the user asks to have displayed. Each subscription expresses an interest in events related to a given stock at the relevant information provider. It then receives all the information sent to it in notifications and displays it to the user. The communication of notifications is illustrated in Figure 6.7.

Figure 6.7 Dealing room system



Characteristics of publish-subscribe systems • Publish-subscribe systems have two main characteristics:

Heterogeneity: When event notifications are used as a means of communication, components in a distributed system that were not designed to interoperate can be made to work together. All that is required is that event-generating objects publish the types of events they offer, and that other objects subscribe to patterns of events and provide an interface for receiving and dealing with the resultant notifications. For example, Bates *et al.* [1996] describe how publish-subscribe systems can be used to connect heterogeneous components in the Internet. They describe a system in which applications can be made aware of users' locations and activities, such as using computers, printers or electronically tagged books. They envisage its future use in the context of a home network supporting commands such as: 'if the children come home, turn on the central heating'.

Asynchronicity: Notifications are sent asynchronously by event-generating publishers to all the subscribers that have expressed an interest in them to prevent publishers needing to synchronize with subscribers – publishers and subscribers need to be decoupled. Mushroom [Kindberg *et al.* 1996] is an object-based publish-subscribe system designed to support collaborative work, in which the user interface displays objects representing users and information objects such as documents and notepads within shared workspaces called *network places*. The state of each place is replicated at the computers of users currently in that place. Events are used to describe changes to objects and to a user's focus of interest. For example, an event

could specify that a particular user has entered or left a place or has performed a particular action on an object. Each replica of any object to which particular types of events are relevant expresses an interest in them through a subscription and receives notifications when they occur. But subscribers to events are decoupled from objects experiencing events, because different users are active at different times.

In addition, a variety of different *delivery guarantees* can be provided for notifications – the one that is chosen should depend on the application requirements. For example, if IP multicast is used to send notifications to a group of receivers, the failure model will relate to the one described for IP multicast in Section 4.4.1 and will not guarantee that any particular recipient will receive a particular notification message. This is adequate for some applications – for example, to deliver the latest state of a player in an Internet game – because the next update is likely to get through.

However, other applications have stronger requirements. Consider the dealing room application: to be fair to the dealers interested in a particular stock, we require that all the dealers for the same stock receive the same information. This implies that a reliable multicast protocol should be used.

In the Mushroom system mentioned above, notifications about changes in object state are delivered reliably to a server, whose responsibility it is to maintain up-to-date copies of objects. However, notifications may also be sent to object replicas in users' computers by means of unreliable multicast; in the case that the latter lose notifications, they can retrieve an object's state from the server. When the application requires it, notifications may be ordered and sent reliably to object replicas.

Some applications have real-time requirements.These include events in a nuclear power station or a hospital patient monitor. It is possible to design multicast protocols that provide real-time guarantees as well as reliability and ordering in a system that satisfies the properties of a synchronous distributed system.

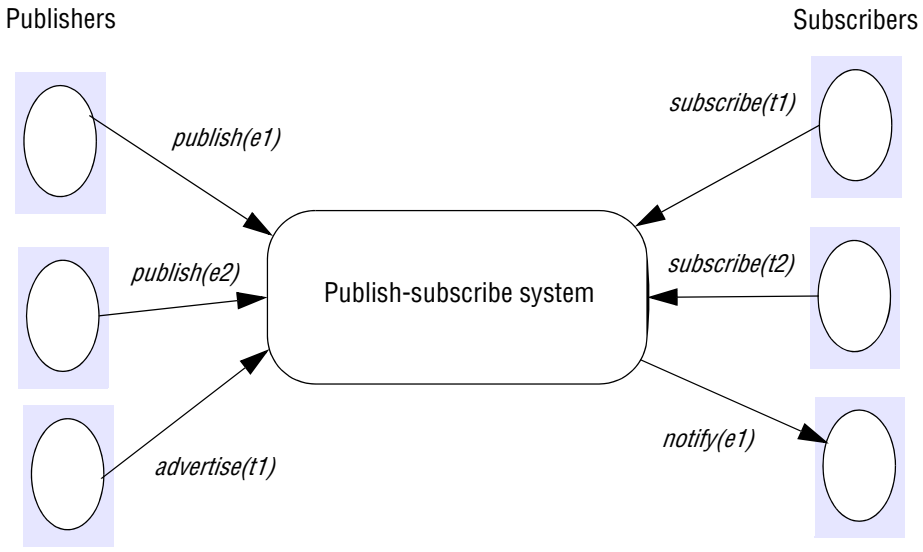
We discuss publish-subscribe systems in more detail in the following sections, considering the programming model they offer before examining some of the key implementation challenges, particularly related to large-scale dissemination of events in the Internet.

6.3.1 The programming model

The programming model in publish-subscribe systems is based on a small set of operations, captured in Figure 6.8. Publishers disseminate an event *e* through a *publish(e)* operation and subscribers express an interest in a set of events through subscriptions. In particular, they achieve this through a *subscribe(f)* operation where *f* refers to a filter – that is, a pattern defined over the set of all possible events. The expressiveness of filters (and hence of subscriptions) is determined by the subscription model; which we discuss in more detail below. Subscribers can later revoke this interest through a corresponding *unsubscribe(f)* operation. When events arrive at a subscriber, the events are delivered using a *notify(e)* operation.

Some systems complement the above set of operations by introducing the concept of advertisements. With advertisements, publishers have the option of declaring the nature of future events through an *advertise(f)* operation. The advertisements are defined in terms of the types of events of interest (these happen to take the same form as filters).

Figure 6.8 The publish-subscribe paradigm



In other words, subscribers declare their interests in terms of subscriptions and publishers optionally declare the styles of events they will generate through advertisements. Advertisements can be revoked through a call of *unadvertise(f)*.

As mentioned above, the expressiveness of publish-subscribe systems is determined by the subscription (filter) model, with a number of schemes defined and considered here in increasing order of sophistication:

Channel-based: In this approach, publishers publish events to named channels and subscribers then subscribe to one of these named channels to receive all events sent to that channel. This is a rather primitive scheme and the only one that defines a physical channel; all other schemes employ some form of filtering over the content of events as we shall see below. Although simple, this scheme has been used successfully in the CORBA Event Service (see Chapter 8).

Topic-based (also referred to as subject-based): In this approach, we make the assumption that each notification is expressed in terms of a number of fields, with one field denoting the topic. Subscriptions are then defined in terms of the topic of interest. This approach is equivalent to channel-based approaches, with the difference that topics are implicitly defined in the case of channels but explicitly declared as one of the fields in topic-based approaches. The expressiveness of topic-based approaches can also be enhanced by introducing hierarchical organization of topics. For example, let us consider a publish-subscribe system for this book. Subscriptions could be defined in terms of *indirect_communication* or *indirect_communication/publish-subscribe*. Subscribers expressing interest in the former will receive all events related to this chapter, whereas with the latter subscribers can instead express an interest in the more specific topic of publish-subscribe.

Content-based: Content-based approaches are a generalization of topic-based approaches allowing the expression of subscriptions over a range of fields in an event notification. More specifically, a content-based filter is a query defined in terms of compositions of constraints over the values of event attributes. For example, a subscriber could express interest in events that relate to the topic of publish-subscribe systems, where the system in question is the 'CORBA Event Service' and where the author is 'Tim Kindberg' or 'Gordon Blair'. The sophistication of the associated query languages varies from system to system, but in general this approach is significantly more expressive than channel- or topic-based approaches, but with significant new implementation challenges (discussed below).

Type-based: These approaches are intrinsically linked with object-based approaches where objects have a specified type. In type-based approaches, subscriptions are defined in terms of types of events and matching is defined in terms of types or subtypes of the given filter. This approach can express a range of filters, from coarse-grained filtering based on overall type names to more fine-grained queries defining attributes and methods of a given object. Such fine-grained filters are similar in expressiveness to content-based approaches. The advantages of type-based approaches are that they can be integrated elegantly into programming languages and they can check the type correctness of subscriptions, eliminating some kinds of subscription errors.

As well as these classical categories, a number of commercial systems are based on subscribing directly to *objects of interest*. Such systems are similar to type-based approaches in that they are intrinsically linked to object-based approaches, although they differ by focusing on changes of state of the objects of interest rather than predicates associated with the types of objects. They allow one object to react to a change occurring in another object. Notifications of events are asynchronous and determined by their receivers. In particular, in interactive applications, the actions that the user performs on objects – for example, by manipulating a button with the mouse or entering text in a text box via the keyboard – are seen as events that cause changes in the objects that maintain the state of the application. The objects that are responsible for displaying a view of the current state are notified whenever the state changes.

Rosenblum and Wolf [1997] describe a general architecture for this style of publish-subscribe system. The main component in their architecture is an event service that maintains a database of event notifications and of interests of subscribers. The event service is notified of events that occur at objects of intetest. Subscribers inform the event service about the types of events they are interested in. When an event occurs at an object of interest a message containing the notification is sent directly to the subscribers of that type of event.

The Jini distributed event specification described by Arnold *et al.* [1999] is one leading example of this approach, and a case study on Jini, together with further background information on this style of approach, can be found on the companion web site for the book [www.cdk5.net/rmi]. Note, however, that Jini is a relatively primitive example of a distributed event-based system that allows direct connectivity between producers and consumers of events (hence compromising time and space uncoupling).

A number of more experimental approaches are also being investigated. For example, some researchers are considering the added expressiveness of *context* [Frey

and Roman 2007, Meier and Cahill 2010]. Context and context-awareness are major concepts in mobile and ubiquitous computing. Context is defined in Chapter 19 as an aspect of the physical circumstances of relevance to the system behaviour. One intuitive example of context is location, and such systems have the potential for users to subscribe to events associated with a given location – for example, any emergency messages associated with the building where a user is located. Cilia *et al*. [2004] have also introduced *concept-based* subscription models whereby filters are expressed in terms of the semantics as well as the syntax of events. More specifically, data items have an associated semantic context that captures the meaning of those items, allowing for interpretation and possible translation into different data formats, thus addressing heterogeneity.

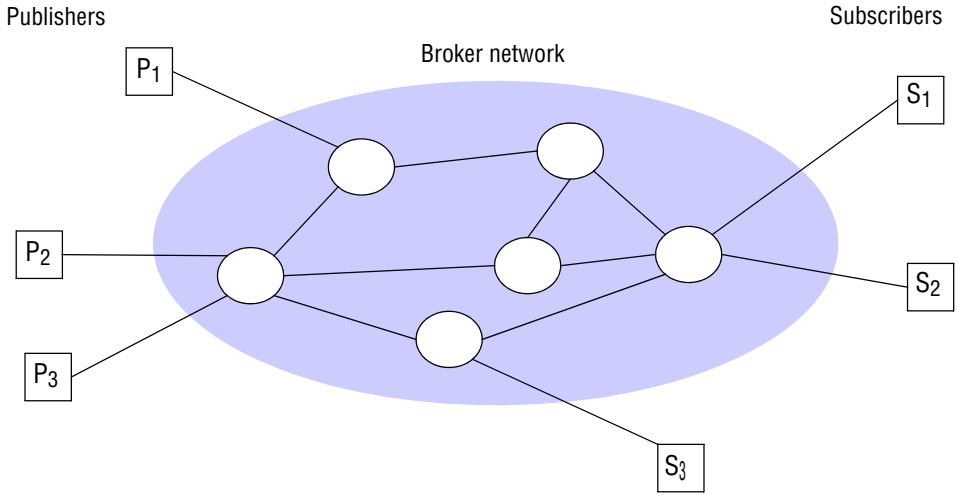
For some classes of application, such as the financial trading system described in Chapter 1, it is not enough for subscriptions to express queries over individual events. Rather, there is a need for more complex systems that can recognize complex event patterns. For example, Chapter 1 introduced the example of buying and selling shares based on observing temporal sequences of events related to share prices, demonstrating the need for *complex event processing* (or composite event detection, as it is sometimes called). Complex event processing allows the specification of patterns of events as they occur in the distributed environment – for example, 'inform me if water levels rise by at least 20% in the River Eden in at least three places and simulation models are also reporting a risk of flooding'. A further example of an event pattern arose in Chapter 1, concerned with detecting share price movements over a given time period. In general, patterns can be logical, temporal or spatial. For further information on complex event processing, refer to Muhl *et al*. [2006].

6.3.2 Implementation issues

From the description above, the task of a publish-subscribe system is clear: to ensure that events are delivered efficiently to all subscribers that have filters defined that match the event. Added to this, there may be additional requirements in terms of security, scalability, failure handling, concurrency and quality of service. This makes the implementation of publish-subscribe systems rather complex, and this has been an area of intense investigation in the research community. We consider key implementation issues below, examining centralized versus distributed implementations before moving on to consider the overall system architecture required to implement publish-subscribe systems (particularly distributed implementations of content-based approaches). We conclude the section by summarizing the design space of publish-subscribe systems, with associated pointers to the literature.

Centralized versus distributed implementations • A number of architectures for the implementation of publish-subscribe systems have been identified. The simplest approach is to centralize the implementation in a single node with a server on that node acting as an event broker. Publishers then publish events (and optionally send advertisements) to this broker, and subscribers send subscriptions to the broker and receive notifications in return. Interaction with the broker is then through a series of point-to-point messages; this can be implemented using message passing or remote invocation.

Figure 6.9 A network of brokers



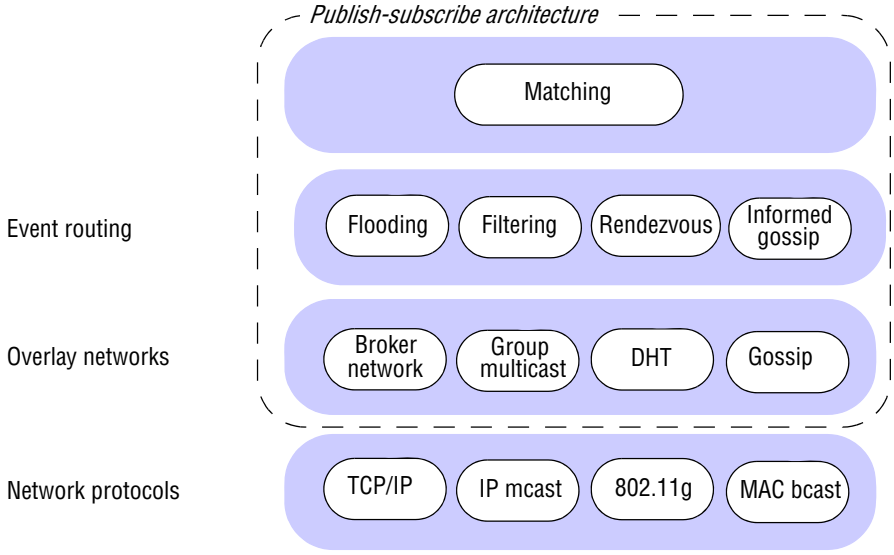
This approach is straightforward to implement, but the design lacks resilience and scalability, since the centralized broker represents a single point for potential system failure and a performance bottleneck. Consequently, distributed implementations of publish-subscribe systems are also available. In such schemes, the centralized broker is replaced by a *network of brokers* that cooperate to offer the desired functionality as illustrated in Figure 6.9. Such approaches have the potential to survive node failure and have been shown to be able to operate well in Internet-scale deployments.

Taking this a step further, it is possible to have a fully *peer-to-peer* implementation of a publish-subscribe system. This is a very popular implementation strategy for recent systems. In this approach, there is no distinction between publishers, subscribers and brokers; all nodes act as brokers, cooperatively implementing the required event routing functionality (as discussed further below).

Overall systems architecture • As mentioned above, the implementation of centralized schemes is relatively straightforward, with the central service maintaining a repository of subscriptions and matching event notifications with this set of subscriptions. Similarly, the implementations of channel-based or topic-based schemes are relatively straightforward. For example, a distributed implementation can be achieved by mapping channels or topics onto associated groups (as defined in Section 6.2) and then using the underlying multicast communication facilities to deliver events to interested parties (using reliable and ordered variants, as appropriate). The distributed implementation of content-based (or by extrapolation, type-based) approaches is more complex and deserving of further consideration. The range of architectural choices for such approaches is captured in Figure 6.10 (adapted from Baldoni and Virgillito [2005]).

In the bottom layer, publish-subscribe systems make use of a range of interprocess communication services, such as TCP/IP, IP multicast (where available) or more specialized services, as offered for example by wireless networks. The heart of the

Figure 6.10 The architecture of publish-subscribe systems



architecture is provided by the event routing layer supported by a network overlay infrastructure. Event routing performs the task of ensuring that event notifications are routed as efficiently as possible to appropriate subscribers, whereas the overlay infrastructure supports this by setting up appropriate networks of brokers or peer-to-peer structures. For content-based approaches, this problem is referred to as *content-based routing* (CBR), with the goal being to exploit content information to efficiently route events to their required destination. The top layer implements matching – that is, ensuring that events match a given subscription. While this can be implemented as a discrete layer, often matching is pushed down into the event routing mechanisms, as will become apparent shortly.

Within this overall architecture, there is a wide variety of implementation approaches. We step through a select set of implementations to illustrate the general principles behind content-based routing:

Flooding: The simplest approach is based on *flooding*, that is, sending an event notification to all nodes in the network and then carrying out the appropriate matching at the subsciber end. As an alternative, flooding can be used to send subscriptions back to all possible publishers, with the matching carried out at the publishing end and matched events sent directly to the relevant subscribers using point-to-point communication. Flooding can be implemented using an underlying broadcast or multicast facility. Alternatively, brokers can be arranged in an acyclic graph in which each forwards incoming event notifications to all its neighbours (effectively providing a multicast overlay, as discussed in Section 4.5.1). This approach has the benefit of simplicity but can result in a lot of unnecessary network

Figure 6.11 Filtering-based routing

```
upon receive publish(event e) from node x                    1
    matchlist := match(e, subscriptions)                     2
    send notify(e) to matchlist;                             3
    fwdlist := match(e, routing);                            4
    send publish(e) to fwdlist - x;                          5
upon receive subscribe(subscription s) from node x           6
    if x is client then                                      7
        add x to subscriptions;                              8
    else add(x, s) to routing;                               9
    send subscribe(s) to neighbours - x;                     10
```

traffic. Hence, the alternative schemes described below try to optimize the number of messages exchanged through consideration of content.

Filtering: One principle that underpins many approaches is to apply *filtering* in the network of brokers. This is referred to as *filtering-based routing*. Brokers forward notifications through the network only where there is a path to a valid subscriber. This is achieved by propagating subscription information through the network towards potential publishers and then storing associated state at each broker. More specifically, each node must maintain a *neighbours list* containing a list of all connected neighbours in the network of brokers, a subscription list containing a list of all directly connected subscribers serviced by this node, and a routing table. Crucially, this routing table maintains a list of neighbours and valid subscriptions for that pathway.

This approach also requires an implementation of matching on each node in the network of brokers: in particular, a *match* function takes a given event notification and a list of nodes together with associated subscriptions and returns a set of nodes where the notification matches the subscription. The specific algorithm for this filtering approach is captured in Figure 6.11 (taken from Baldoni and Virgillito [2005]). When a broker receives a publish request from a given node, it must pass this notification to all connected nodes where there is a corresponding matching subscription and also decide where to propagate this event through the network of brokers. Lines 2 and 3 achieve the first goal by matching the event against the subscription list and then forwarding the event to all the nodes with matching subscriptions (the *matchlist*). Lines 4 and 5 then use the match function again, this time matching the event against the routing table and forwarding only to the paths that lead to a subscription (the *fwdlist*). Brokers must also deal with incoming subscription events. If the subscription event is from an immediately connected subscriber, then this subscription must be entered in the subscriptions table (lines 7 and 8). Otherwise, the broker is an intermediary node; this node now knows that a pathway exists towards this subscription and hence an appropriate entry is added to the routing table (line 9). In both cases, this subscription event is then passed to all neighbours apart from the originating node (line 10).

Figure 6.12 Rendezvous-based routing

```
upon receive publish(event e) from node x at node i
    rvlist := EN(e);
    if i in rvlist then begin
        matchlist <- match(e, subscriptions);
        send notify(e) to matchlist;
    end
    send publish(e) to rvlist - i;
upon receive subscribe(subscription s) from node x at node i
    rvlist := SN(s);
    if i in rvlist then
        add s to subscriptions;
    else
        send subscribe(s) to rvlist - i;
```

Advertisements: The pure filtering-based approach described above can generate a lot of traffic due to propagation of subscriptions, with subscriptions essentially using a flooding approach back towards all possible publishers. In systems with *advertisements* this burden can be reduced by propagating the advertisements towards subscribers in a similar (actually, symmetrical) way to the propagation of subscriptions. There are interesting trade-offs between the two approaches, and some systems adopt both approaches in tandem [Carzaniga *et al.* 2001].

Rendezvous: Another approach to control the propagation of subscriptions (and to achieve a natural load balancing) is the *rendezvous* approach. To understand this approach, it is necessary to view the set of all possible events as an event space and to partition responsibility for this event space between the set of brokers in the network. In particular, this approach defines rendezvous nodes, which are broker nodes responsible for a given subset of the event space. To achieve this, a given *rendezvous-based routing* algorithm must define two functions. First, *SN(s)* takes a given subscription, *s*, and returns one or more rendezvous nodes that take responsibility for that subscription. Each such rendezvous node maintains a subscription list as in the filtering approach above, and forwards all matching events to the set of subscribing nodes. Second, when an event *e* is published, the function *EN(e)* also returns one or more rendezvous nodes, this time responsible for matching *e* against subscriptions in the system. Note that both *SN(s)* and *EN(e)* return more than one node if reliability is a concern. Note also that this approach only works if the intersection of *EN(e)* and *SN(s)* is non-empty for a given *e* that matches *s* (known as the mapping intersection rule, as defined by Baldoni and Virgillito [2005]). The corresponding code for rendezvous-based routing is shown in Figure 6.12 (again taken from Baldoni and Virgillito [2005]). This time, we leave the interpretation of the algorithm as an exercise for the reader (see Exercise 6.11).

One interesting interpretation of rendezvous-based routing is to map the event space onto a *distributed hash table* (DHT). Distributed hash tables were introduced briefly in Section 4.5.1 and are examined in more detail in Chapter 10. A distributed

Figure 6.13 Example publish-subscribe systems

System (and further reading)	*Subscription model*	*Distribution model*	*Event routing*
CORBA Event Service (Chapter 8)	Channel-based	Centralized	-
TIB Rendezvouz [Oki *et al.* 1993]	Topic-based	Distributed	Ffiltering
Scribe [Castro *et al.* 2002b]	Topic-based	Peer-to-peer (DHT)	Rendezvous
TERA [Baldoni *et al.* 2007]	Topic-based	Peer-to-peer	Informed gossip
Siena [Carzaniga *et al.* 2001]	Content-based	Distributed	Filtering
Gryphon [www.research.ibm.com]	Content-based	Distributed	Filtering
Hermes [Pietzuch and Bacon 2002]	Topic- and content-based	Distributed	Rendezvous and filtering
MEDYM [Cao and Singh 2005]	Content-based	Distributed	Flooding
Meghdoot [Gupta *et al.* 2004]	Content-based	Peer-to-peer	Rendezvous
Structure-less CBR [Baldoni *et al.* 2005]	Content-based	Peer-to-peer	Informed gossip

hash table is a style of network overlay that distributes a hash table over a set of nodes in a peer-to-peer network. The key observation for rendezvous-based routing is that the hash function can be used to map both events and subscriptions onto a corresponding rendezvous node for the management of such subscriptions.

It is possible to employ other peer-to-peer middleware approaches to underpin event routing in publish-subscribe systems. Indeed, this is a very active area of research with many novel and interesting proposals emerging, particularly for very large-scale systems [Carzaniga *et al.* 2001]. One specific approach is to adopt *gossiping* as a means of supporting event routing. Gossip-based approaches are a popular mechanism for achieving multicast (including reliable multicast), as discussed in Section 18.4.1. They operate by nodes in the network periodically and probabilistically exchanging events (or data) with neighbouring nodes. Through this approach, it is possible to propagate events effectively through the network without the structure imposed by other approaches. A pure gossip approach is effectively an alternative strategy for implementing flooding, as described above. However, it is possible to take into account local information and, in particular, content to achieve what is referred to as *informed gossip*. Such approaches can be particularly attractive in highly dynamic environments where network or node churn can be high [Baldoni *et al*. 2005].

6.3.3 Examples of publish-subscribe systems

We conclude this section by listing some major examples of publish-subscribe systems, providing references for further reading (see Figure 6.13). This figure also captures the design space for publish-subscribe systems, illustrating how different designs can result from decisions on subscription and distribution models and, especially, the underlying event routing strategy. Note that event routing is not required for centralized schemes, hence the blank entry in the table.

6.4 Message queues

Message queues (or more accurately, distributed message queues) are a further important category of indirect communication systems. Whereas groups and publish-subscribe provide a one-to-many style of communication, message queues provide a *point-to-point* service using the concept of a message queue as an indirection, thus achieving the desired properties of space and time uncoupling. They are point-to-point in that the sender places the message into a queue, and it is then removed by a single process. Message queues are also referred to as Message-Oriented Middleware. This is a major class of commercial middleware with key implementations including IBM's WebSphere MQ, Microsoft's MSMQ and Oracle's Streams Advanced Queuing (AQ). The main use of such products is to achieve *Enterprise Application Integration* (EAI) – that is, integration between applications within a given enterprise – a goal that is achieved by the inherent loose coupling of message queues. They are also extensively used as the basis for *commercial transaction processing systems* because of their intrinsic support for transactions, discussed further in Section 6.4.1.

We examine message queues in more detail below, considering the programming model offered by message queueing systems before addressing implementation issues. The section then concludes by presenting the Java Messaging Service (JMS) as an example of a middleware specification supporting message queues (and also publish-subscribe).

6.4.1 The programming model

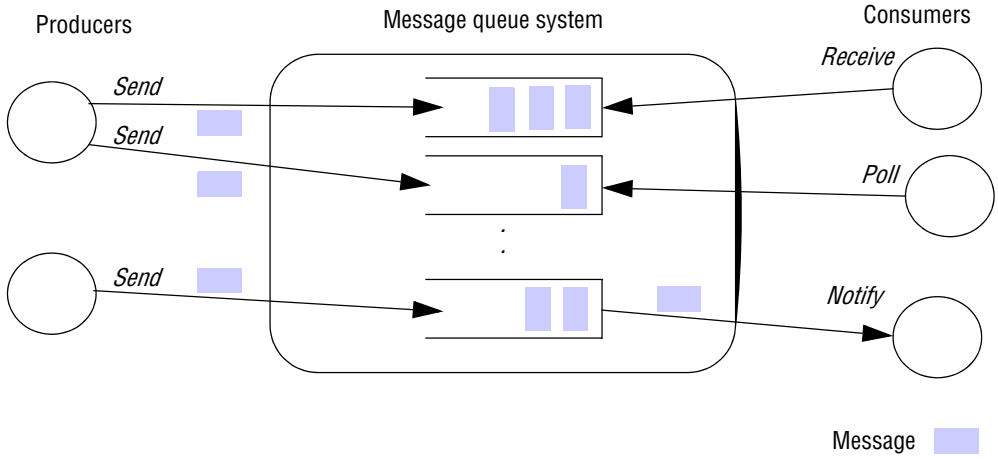
The programming model offered by message queues is very simple. It offers an approach to communication in distributed systems through queues. In particular, producer processes can *send* messages to a specific queue and other (consumer) processes can then receive messages from this queue. Three styles of receive are generally supported:

- a *blocking receive*, which will block until an appropriate message is available;
- a *non-blocking receive* (a polling operation), which will check the status of the queue and return a message if available, or a not available indication otherwise;
- a *notify* operation, which will issue an event notification when a message is available in the associated queue.

This overall approach is captured pictorially in Figure 6.14.

A number of processes can send messages to the same queue, and likewise a number of receivers can remove messages from a queue. The queuing policy is normally *first-in-first-out* (FIFO), but most message queue implementations also support the concept of priority, with higher-priority messages delivered first. Consumer processes can also *select* messages from the queue based on properties of a message. In more detail, a message consists of a *destination* (that is, a unique identifier designating the destination queue), *metadata* associated with the message, including fields such as the priority of the message and the delivery mode, and also the *body* of the message. The body is normally opaque and untouched by the message queue system. The associated

Figure 6.14 The message queue paradigm



content is serialized using any of the standard approaches described in Section 4.3; that is, marshalled data types, object serialization or XML structured messages. Message sizes are configurable and can be very large – for example, on the order of a 100 Mbytes Given the fact that message bodies are opaque, message selection is normally expressed through predicates defined over the metadata.

Oracle's AQ introduces an interesting twist on this basic idea to achieve better integration with (relational) databases; in Oracle AQ, messages are rows in a database table, and queues are database tables that can be queried using the full power of a database query language.

One crucial property of message queue systems is that messages are *persistent* – that is, message queues will store the messages indefinitely (until they are consumed) and will also commit the messages to disk to enable *reliable delivery*. In particular, following the definition of reliable communication in Section 2.4.2, any message sent is eventually received (validity) and the message received is identical to the one sent, and no messages are delivered twice (integrity). Message queue systems therefore guarantee that messages will be delivered (and delivered once) but cannot say anything about the timing of the delivery.

Message passing systems can also support additional functionality:

- Most commercially available systems provide support for the sending or receiving of a message to be contained within a *transaction*. The goal is to ensure that all the steps in the transaction are completed, or the transaction has no effect at all (the 'all or nothing' property). This relies on interfacing with an external transaction service, provided by the middleware environment. Detailed consideration of transactions is deferred until Chapter 16.

- A number of systems also support message transformation, whereby an arbitrary transformation can be performed on an arriving message. The most common application of this concept is to transform messages between formats to deal with heterogeneity in underlying data representations. This could be as simple as

transforming from one byte order to another (big-endian to little-endian) or more complex, involving for example a transformation from one external data representation to another (such as SOAP to IIOP). Some systems also allow programmers to develop their own application-specific transformation in response to triggers from the underlying message queuing system. Message transformation is an important tool in dealing with heterogeneity generally and achieving Enterprise Application Integration in particular (as discussed above). Note that the term *message broker* is often used to denote a service responsible for message transformation.

- Some message queue implementations also provide support for *security*. For example, WebSphere MQ provides support for the confidential transmission of data using the Secure Sockets Layer (SSL) together with support for authentication and access control. See Chapter 11.

As a final word on the programming abstraction offered by message queues, it is helpful to compare the style of programming with other communication paradigms. Message queues are similar in many ways to the message-passing systems considered in Chapter 4. The difference is that whereas message-passing systems have implicit queues associated with senders and receivers (for example, the message buffers in MPI), message queuing systems have explicit queues that are third-party entities, separate from the sender and the receiver. It is this key difference that makes message queues an indirect communication paradigm with the crucial properties of space and time uncoupling.

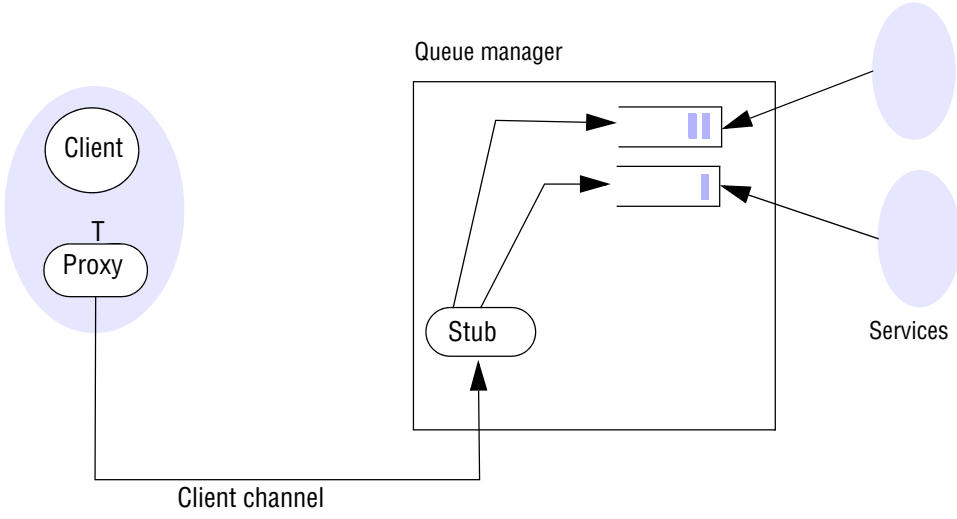
6.4.2 Implementation issues

The key implementation issue for message queuing systems is the choice between centralized and distributed implementations of the concept. Some implementations are centralized, with one or more message queues managed by a queue manager located at a given node. The advantage of this scheme is simplicity, but such managers can become rather heavyweight components and have the potential to become a bottleneck or a single point of failure. As a result, more distributed implementations have been proposed. To illustrate distributed architectures, we briefly consider the approach adopted in WebSphere MQ as representative of the state-of-the-art in this area.

Case study: WebSphere MQ • WebSphere MQ is middleware developed by IBM based on the concept of message queues, offering an indirection between senders and receivers of messages [www.redbooks.ibm.com]. Queues in WebSphere MQ are managed by *queue managers* which host and manage queues and allow applications to access queues through the *Message Queue Interface* (MQI). The MQI is a relatively simple interface allowing applications to carry out operations such as connecting to or disconnecting from a queue (*MQCONN* and *MQDISC*) or sending/receiving messages to/from a queue (*MQPUT* and *MQGET*). Multiple queue managers can reside on a single physical server.

Client applications accessing a queue manager may reside on the same physical server. More generally, though, they will be on different machines and must then communicate with the queue manager through what is known as a *client channel*. Client channels adopt the rather familiar concept of a proxy, as introduced in Chapters 2 and 5,

Figure 6.15 A simple networked topology in WebSphere MQ



whereby MQI commands are issued on the proxy and then sent transparently to the queue manager for execution using RPC. An example of such a configuration is shown in Figure 6.15. In this configuration, a client application is sending messages to a remote queue manager and multiple services (on the same machine as the server) are then consuming the incoming messages.

This is a very simple use of WebSphere MQ, and in practice it is more common for queue managers to be linked together into a federated structure, mirroring the approach often adopted in publish-subscribe systems (with networks of brokers). To achieve this, MQ introduces the concept of a *message channel* as a unidirectional connection between two queue managers that is used to forward messages asynchronously from one queue to another. Note the terminology here: a message channel is a connection between two queue managers, whereas a client channel is a connection between a client application and a queue manager. A message channel is managed by a *message channel agent* (MCA) at each end. The two agents are responsible for establishing and maintaining the channel, including an initial negotiation to agree on the properties of the channel (including security properties). Routing tables are also included in each queue manager, and together with channels this allows arbitrary topologies to be created.

This ability to create customized topologies is crucial to WebSphere MQ, allowing users to determine the right topology for their application domain, for example to deliver certain requirements in terms of scalability and performance. Tools are provided for systems administrators to create suitable topologies and to hide the complexities of establishing message channels and routing strategies.

A wide range of topologies can be created, including trees, meshes or a bus-based configuration. To illustrate the concept of topologies further, we present one example topology often used in WebSphere MQ deployments, the hub-and-spoke topology.

The hub-and-spoke approach: In the hub-and-spoke topology, one queue manager is designated as the hub. The hub hosts a range of services. Client applications do not connect directly to this hub but rather connect through queue managers designated as spokes. Spokes relay messages to the message queue of the hub for processing by the various services. Spokes are placed strategically around the network to support different clients. The hub is placed somewhere appropriate in the network, on a node with sufficient resources to deal with the volume of traffic. Most applications and services are located on the hub, although it is also possible to have some more local services on spokes.

This topology is heavily used with WebSphere MQ, particularly in large-scale deployments covering significant geographical areas (and possibly crossing organizational boundaries). The key to the approach is to be able to connect to a local spoke over a high-bandwidth connection, for example over a local area network (spokes may even be placed in the same physical machine as client applications to minimize latency).

Recall that communication between a client application and a queue manager uses RPC, whereas internal communication between queue managers is asynchronous (non-blocking). This means that the client application is only blocked until the message is deposited in the local queue manager (the local spoke); subsequent delivery, potentially over wide area networks, is asynchronous but guaranteed to be reliable by the WebSphere MQ middleware.

Clearly, the drawback of this architecture is that the hub can be a potential bottleneck and a single point of failure. WebSphere MQ also supports other facilities to overcome these problems, including queue manager clusters, which allow multiple instances of the same service to be supported by multiple queue managers with implicit load balancing across the different instantiations [www.redbooks.ibm.com].

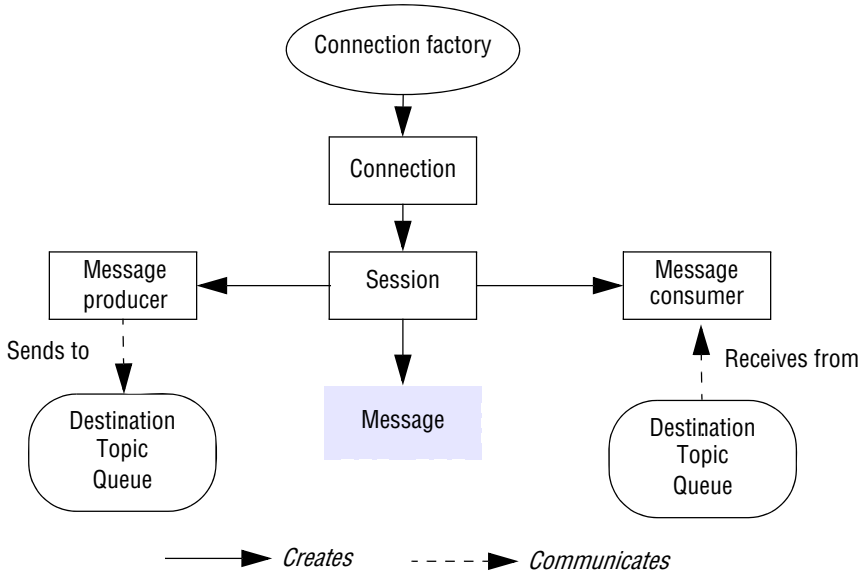
6.4.3 Case study: The Java Messaging Service (JMS)

The *Java Messaging Service* (JMS) [java.sun.com XI] is a specification of a standardized way for distributed Java programs to communicate indirectly. Most notably, as will be explained, the specification unifies the publish-subscribe and message queue paradigms at least superficially by supporting topics and queues as alternative destinations of messages. A wide variety of implementations of the common specification are now available, including Joram from OW2, Java Messaging from JBoss, Sun's Open MQ, Apache ActiveMQ and OpenJMS. Other platforms, including WebSphere MQ, also provide a JMS interface on to their underlying infrastructure.

JMS distinguishes between the following key roles:

- A JMS client is a Java program or component that produces or consumes messages, a JMS producer is a program that creates and produces messages and a JMS consumer is a program that receives and consumes messages.
- A JMS provider is any of the multiple systems that implement the JMS specification.
- A JMS message is an object that is used to communicate information between JMS clients (from producers to consumers).

Figure 6.16 The programming model offered by JMS



- A JMS destination is an object supporting indirect communication in JMS. It is either a JMS topic or a JMS queue.

Programming with JMS • The programming model offered by the JMS API is captured in Figure 6.16. To interact with a JMS provider, it is first necessary to create a *connection* between a client program and the provider. This is created through a *connection factory* (a service responsible for creating connections with the required properties). The resultant connection is a logical channel between the client and provider; the underlying implementation may, for example, map onto a TCP/IP socket if implemented over the Internet. Note that two types of connection can be established, a *TopicConnection* or a *QueueConnection*, thus enforcing a clear separation between the two modes of operation within given connections.

Connections can be used to create one or more *sessions* – a session is a series of operations involving the creation, production and consumption of messages related to a logical task. The resultant session object also supports operations to create *transactions*, supporting all-or-nothing execution of a series of operations, as discussed in Section 6.4.1. There is a clear distinction between topic sessions and queue sessions in that a *TopicConnection* can support one or more topic sessions and a *QueueConnection* can support one or more queue sessions, but it is not possible to mix session styles in a connection. Thus, the two styles of operation are integrated in a rather superficial way.

The session object is central to the operation of JMS, supporting methods for the creation of messages, message producers and message consumers:

- In JMS, a *message* consists of three parts: a *header*, a set of *properties* and the *body* of the message. The header contains all the information needed to identify and route the message, including the destination (a reference to either a topic or a

Figure 6.17 Java class *FireAlarmJMS*

```
import javax.jms.*;
import javax.naming.*;

public class FireAlarmJMS {

    public void raise() {
    try {                                                                    1
        Context ctx = new InitialContext();                                  2
        TopicConnectionFactory topicFactory =                                3
            (TopicConnectionFactory)ctx.lookup("TopicConnectionFactory");    4
        Topic topic = (Topic)ctx.lookup("Alarms");                           5
        TopicConnection topicConn =                                          6
            topicConnectionFactory.createTopicConnection();                  7
        TopicSession topicSess = topicConn.createTopicSession(false,         8
                                          Session.AUTO_ACKNOWLEDGE);         9
        TopicPublisher topicPub = topicSess.createPublisher(topic);          10
        TextMessage msg = topicSess.createTextMessage();                     11
        msg.setText("Fire!");                                                12
        topicPub.publish(message);                                           13
        } catch (Exception e) {                                              14
    }                                                                        15
}
```

queue), the priority of the message, the expiration date, a message ID and a timestamp. Most of these fields are created by the underlying system, but some can be filled in specifically through the associated constructor methods. Properties are all user-defined and can be used to associate other application-specific metadata elements with a message. For example, if implementing a context-aware system (as discussed in Chapter 19), the properties can be used to express additional context associated with the message, including a location field. As in the general description of message queue systems, this body is opaque and untouched by the system. In JMS, the body can be any one of a text message, a byte stream, a serialized Java object, a stream of primitive Java values or a more structured set of name/value pairs.

- A *message producer* is an object used to publish messages under a particular topic or to send messages to a queue.
- A *message consumer* is an object used to subscribe to messages concerned with a given topic or to receive messages from a queue. The consumer is more complicated than the producer, for two reasons. First, it is possible to associate filters with message consumers by specifying what is known as a *message selector* – a predicate defined over the values in the header and properties parts of a message (not the body). A subset of the database query language SQL is used to specify properties. This could be used, for example, to filter messages from a

Figure 6.18 Java class *FireAlarmConsumerJMS*

```
import javax.jms.*;
import javax.naming.*;

public class FireAlarmConsumerJMS {
    public String await() {
    try {                                                                    1
        Context ctx = new InitialContext();                                  2
        TopicConnectionFactory topicFactory =                                3
            (TopicConnectionFactory)ctx.lookup("TopicConnectionFactory");    4
        Topic topic = (Topic)ctx.lookup("Alarms");                           5
        TopicConnection topicConn =                                          6
            topicConnectionFactory.createTopicConnection();                  7
        TopicSession topicSess = topicConn.createTopicSession(false,         8
                                        Session.AUTO_ACKNOWLEDGE);           9
        TopicSubscriber topicSub = topicSess.createSubscriber(topic);        10
        topicSub.start();                                                    11
        TextMessage msg = (TextMessage) topicSub.receive();                  12
        return msg.getText();                                                13
        } catch (Exception e) {                                              14
            return null;                                                     15
    }                                                                        16
}
```

given location in the context-aware example above. Second, there are two modes provided for receiving messages: the program either can block using a *receive* operation or it can establish a *message listener* object which must provide an *onMessage* method that is invoked whenever a suitable message is identified.

A simple example • To illustrate the use of JMS, we return to our example of Section 6.2.3, the fire alarm service, and show how this would be implemented in JMS. We choose the topic-based publish-subscribe service as this is intrinsically a one-to-many application, with the alarm producing alarm messages targeted towards many consumer applications.

The code for the fire alarm object is shown in Figure 6.17. It is more complicated than the equivalent JGroups example mainly because of the need to create a connection, session, publisher and message, as shown in lines 6–11. This is relatively straightforward apart from the parameters of *createTopicSession*, which are whether the session should be transactional (*false* in this case) and the mode of acknowledging messages (*AUTO_ACKNOWLEDGE* in this example, which means a session automatically acknowledges the receipt of a message). There is additional complexity associated with finding the connection factory and topic in the distributed environment (the complexity of connecting to a named channel in JGroups is all hidden in the *connect* method). This is achieved using JNDI (the Java Naming and Directory Interface) in lines 2 to 5. This is included for completeness and it is assumed that readers can appreciate

the purpose of these lines of code without further explanation. Lines 12 and 13 contain the crucial code to create a new message and then publish it to the appropriate topic. The code to create a new instance of the *FireAlarmJMS* class and then raise an alarm is:

```
FireAlarmJMS alarm = new FireAlarmJMS();
alarm.raise();
```

The corresponding code for the receiver end is very similar and is shown in Figure 6.18. Lines 2–9 are identical and create the required connection and session, respectively. This time, though, an object of type *TopicSubscriber* is created next (line 10), and the *start* method in line 11 starts this subscription, enabling messages to be received. The blocking *receive* in line 12 then awaits an incoming message and line 13 returns the textual contents of this message as a string. This class is used as follows by a consumer:

```
FireAlarmConsumerJMS alarmCall = new FireAlarmConsumerJMS();
String msg = alarmCall.await();
System.out.println("Alarm received: "+msg);
```

Overall this case study has illustrated how both publish-subscribe and message queues can be supported by a single middleware solution (in this case JMS), offering the programmer the choice of one-to-many or point-to-point variants of indirect communication, respectively.

6.5 Shared memory approaches

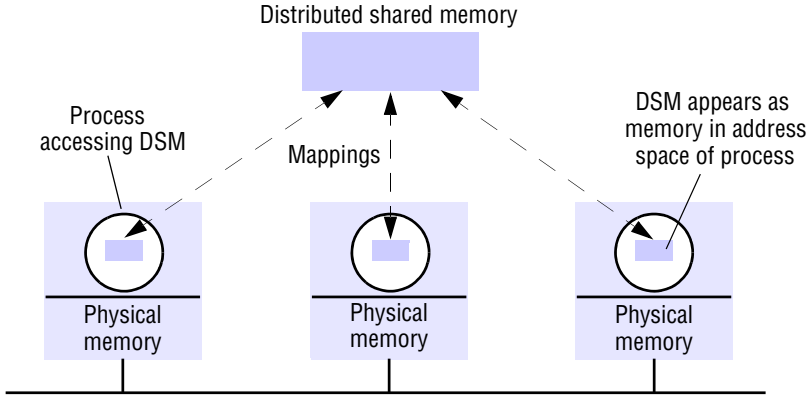
In this section, we examine indirect communication paradigms that offer an abstraction of shared memory. We look briefly at distributed shared memory techniques that were developed principally for parallel computing before moving on to tuple space communication, an approach that allows programmers to read and write tuples from a shared tuple space. Whereas distributed shared memory operates at the level of reading and writing bytes, tuple spaces offer a higher-level perspective in the form of semi-structured data. In addition, whereas distributed shared memory is accessed by address, tuple spaces are *associative*, offering a form of content-addressable memory [Gelernter 1985].

Chapter 18 of the fourth edition of this book provided in-depth coverage of distributed shared memory, including consistency models and several case studies. This chapter can be found on the companion web site for the book [www.cdk5.net/dsm].

6.5.1 Distributed shared memory

Distributed shared memory (DSM) is an abstraction used for sharing data between computers that do not share physical memory. Processes access DSM by reads and updates to what appears to be ordinary memory within their address space. However, an underlying runtime system ensures transparently that processes executing at different computers observe the updates made by one another. It is as though the processes access a single shared memory, but in fact the physical memory is distributed (see Figure 6.19).

Figure 6.19 The distributed shared memory abstraction



The main point of DSM is that it spares the programmer the concerns of message passing when writing applications that might otherwise have to use it. DSM is primarily a tool for parallel applications or for any distributed application or group of applications in which individual shared data items can be accessed directly. DSM is in general less appropriate in client-server systems, where clients normally view server-held resources as abstract data and access them by request (for reasons of modularity and protection).

Message passing cannot be avoided altogether in a distributed system: in the absence of physically shared memory, the DSM runtime support has to send updates in messages between computers. DSM systems manage replicated data: each computer has a local copy of recently accessed data items stored in DSM, for speed of access. The problems of implementing DSM are related to the replication issues to be discussed in Chapter 18, as well as to those of caching shared files, discussed in Chapter 12.

One of the first notable examples of a DSM implementation was the Apollo Domain file system [Leach *et al*. 1983], in which processes hosted by different workstations share files by mapping them simultaneously into their address spaces. This example shows that distributed shared memory can be persistent. That is, it may outlast the execution of any process or group of processes that accesses it and be shared by different groups of processes over time.

The significance of DSM first grew alongside the development of shared-memory multiprocessors (discussed further in Section 7.3). Much research has gone into investigating algorithms suitable for parallel computation on these multiprocessors. At the hardware architectural level, developments include both caching strategies and fast processor-memory interconnections, aimed at maximizing the number of processors that can be sustained while achieving fast memory access latency and throughput [Dubois *et al*. 1988]. Where processes are connected to memory modules over a common bus, the practical limit is on the order of 10 processors before performance degrades drastically due to bus contention. Processors sharing memory are commonly constructed in groups of four, sharing a memory module over a bus on a single circuit board. Multiprocessors with up to 64 processors in total are constructed from such boards in a *Non-Uniform Memory Access* (NUMA) architecture. This is a hierarchical

architecture in which the four-processor boards are connected using a high-performance switch or higher-level bus. In a NUMA architecture, processors see a single address space containing all the memory of all the boards. But the access latency for on-board memory is less than that for a memory module on a different board – hence the name of this architecture.

In *distributed-memory multiprocessors* and clusters of off-the-shelf computing components (again, see Section 7.3), the processors do not share memory but are connected by a very high speed network. These systems, like general-purpose distributed systems, can scale to much greater numbers of processors than a shared-memory multiprocessor's 64 or so. A central question that has been pursued by the DSM and multiprocessor research communities is whether the investment in knowledge of shared memory algorithms and the associated software can be directly transferred to a more scalable distributed memory architecture.

Message passing versus DSM • As a communication mechanism, DSM is comparable with message passing rather than with request-reply-based communication, since its application to parallel processing, in particular, entails the use of asynchronous communication. The DSM and message-passing approaches to programming can be contrasted as follows:

Service offered: Under the message-passing model, variables have to be marshalled from one process, transmitted and unmarshalled into other variables at the receiving process. By contrast, with shared memory the processes involved share variables directly, so no marshalling is necessary – even of pointers to shared variables – and thus no separate communication operations are necessary. Most implementations allow variables stored in DSM to be named and accessed similarly to ordinary unshared variables. In favour of message passing, on the other hand, is that it allows processes to communicate while being protected from one another by having private address spaces, whereas processes sharing DSM can, for example, cause one another to fail by erroneously altering data. Furthermore, when message passing is used between heterogeneous computers, marshalling takes care of differences in data representation; but how can memory be shared between computers with, for example, different integer representations?

Synchronization between processes is achieved in the message model through message passing primitives themselves, using techniques such as the lock server implementation discussed in Chapter 16. In the case of DSM, synchronization is via normal constructs for shared-memory programming such as locks and semaphores (although these require different implementations in the distributed memory environment). Chapter 7 briefly discusses such synchronization objects in the context of programming with threads.

Finally, since DSM can be made persistent, processes communicating via DSM may execute with non-overlapping lifetimes. A process can leave data in an agreed memory location for the other to examine when it runs. By contrast, processes communicating via message passing must execute at the same time.

Efficiency: Experiments show that certain parallel programs developed for DSM can be made to perform about as well as functionally equivalent programs written for message-passing platforms on the same hardware [Carter *et al*. 1991] – at least in the

case of relatively small numbers of computers (10 or so). However, this result cannot be generalized. The performance of a program based on DSM depends upon many factors, as we shall discuss below – particularly the pattern of data sharing (such as whether an item is updated by several processes).

There is a difference in the visibility of costs associated with the two types of programming. In message passing, all remote data accesses are explicit and therefore the programmer is always aware of whether a particular operation is in-process or involves the expense of communication. Using DSM, however, any particular read or update may or may not involve communication by the underlying runtime support. Whether it does or not depends upon such factors as whether the data have been accessed before and the sharing pattern between processes at different computers.

There is no definitive answer as to whether DSM is preferable to message passing for any particular application. DSM remains a tool whose ultimate status depends upon the efficiency with which it can be implemented.

6.5.2 Tuple space communication

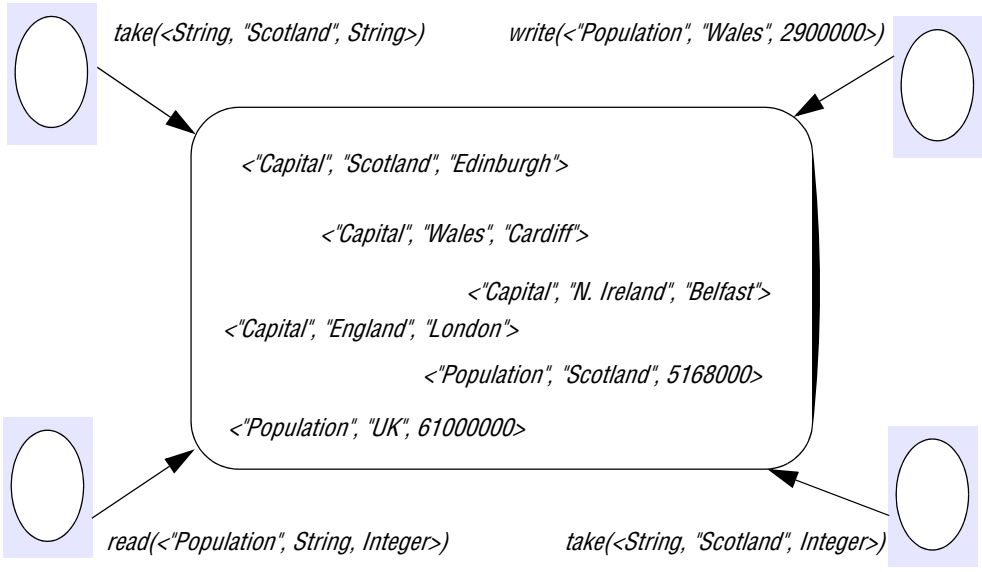
Tuple spaces were first introduced by David Gelernter from Yale University as a novel form of distributed computing based on what he refers to as *generative communication* [Gelernter 1985]. In this approach, processes communicate indirectly by placing tuples in a tuple space, from which other processes can read or remove them. Tuples do not have an address but are accessed by pattern matching on content (content-addressable memory, as discussed by Gelernter [1985]). The resultant Linda programming model has been highly influential and has led to significant developments in distributed programming including systems such as Agora [Bisiani and Forin 1988] and, more significantly, JavaSpaces from Sun (discussed below) and IBM's TSpaces. Tuple space communication has also been influential in the field of ubiquitous computing, for reasons that are explored in depth in Chapter 19.

This section provides an examination of the tuple space paradigm as it applies to distributed computing. We start by examining the programming model offered by tuple spaces before briefly considering the associated implementation issues. The section then concludes by examining the JavaSpaces specification as a case study, illustrating how tuple spaces have evolved to embrace the object-oriented world.

The programming model • In the tuple space programming model, processes communicate through a tuple space – a shared collection of tuples. Tuples in turn consist of a sequence of one or more typed data fields such as <*"fred", 1958*>, <*"sid", 1964*> and <*4, 9.8, "Yes"*>. Any combination of types of tuples may exist in the same tuple space. Processes share data by accessing the same tuple space: they place tuples in tuple space using the *write* operation and read or extract them from tuple space using the *read* or *take* operation. The *write* operation adds a tuple without affecting existing tuples in the space. The *read* operation returns the value of one tuple without affecting the contents of the tuple space. The *take* operation also returns a tuple, but in this case it also removes the tuple from the tuple space.

When reading or removing a tuple from tuple space, a process provides a tuple specification and the tuple space returns any tuple that matches that specification – as mentioned above, this is a type of associative addressing. To enable processes to

Figure 6.20 The tuple space abstraction



synchronize their activities, the read and take operations both block until there is a matching tuple in the tuple space. A tuple specification includes the number of fields and the required values or types of the fields. For example, *take*(<*String*, *integer*>) could extract either <"*fred*", *1958*> or <"*sid*", *1964*>; *take*(<*String*, *1958*>) would extract only <"*fred*", *1958*> of those two.

In the tuple space paradigm, no direct access to tuples in tuple space is allowed and processes have to replace tuples in the tuple space instead of modifying them. Thus, tuples are *immutable*. Suppose, for example, that a set of processes maintains a shared counter in tuple space. The current count (say, 64) is in the tuple <*"counter"*, *64*>. A process must execute code of the following form in order to increment the counter in a tuple space *myTS*:

```
<s, count> := myTS.take(<"counter", integer>);
myTS.write(<"counter", count+1>);
```

A further illustration of the tuple space paradigm is given in Figure 6.20. This tuple space contains a range of tuples representing geographical information about countries in the United Kingdom, including populations and capital cities. The *take* operation *take(<String, "Scotland", String>)* will match <*"Capital", "Scotland", "Edinburgh"*>, whereas *take(<String, "Scotland", Integer>)* will match <*"Population", "Scotland", 5168000*>. The *write* operation *write(<"Population", "Wales, 2900000>)* will insert a new tuple in the tuple space with information on the population of Wales. Finally, *read(<"Population", String, Integer)* can match the equivalent tuples for the populations of the UK, Scotland or indeed Wales, if this operation is executed after the corresponding *write* operation. One will be selected nondeterministically by the tuple

space implementation and, with this being a *read* operation, the tuple will remain in the tuple space.

Note that *write*, *read* and *take* are known as *out*, *rd* and *in* in Linda; we use the more descriptive former names throughout this book. This terminology is also used in JavaSpaces, discussed in a case study below.

Properties associated with tuple spaces: Gelernter [1985] presents some interesting properties associated with tuple space communication, highlighting in particular both space and time uncoupling as discussed in Section 6.1:

> *Space uncoupling*: A tuple placed in tuple space may originate from any number of sender processes and may be delivered to any one of a number of potential recipients. This property is also referred to as *distributed naming* in Linda.
>
> *Time uncoupling*: A tuple placed in tuple space will remain in that tuple space until removed (potentially indefinitely), and hence the sender and receiver do not need to overlap in time.

Together, these features provide an approach that is fully distributed in space and time and also provide for a form of *distributed sharing* of shared variables via the tuple space.

Gelernter [1985] also explores a range of other properties associated with the rather flexible style of naming employed in Linda (referred to as *free naming*). The interested reader is directed to Gelernter's paper for more information on this topic.

Variations on a theme: Since the introduction of Linda, refinements have been proposed to the original model:

- The original Linda model proposed a single, global tuple space. This is not optimal in large systems, as it leads to the danger of unintended aliasing of tuples: as the number of tuples in a tuple space increases, there is an increasing chance of a *read* or *take* matching a tuple by accident. This is particularly likely when matching on types, such as with *take*(<*String, integer*>), as mentioned above. Given this, a number of systems have proposed *multiple tuple spaces*, including the ability to dynamically create tuple spaces, introducing a degree of scoping into the system (see, for example, the JavaSpaces case study below).
- Linda was anticipated to be implemented as a centralized entity but later systems have experimented with *distributed* implementations of tuple spaces (including strategies to provide more fault tolerance). Given the importance of this topic to this book, we focus on this in the implementation issues subsection below.
- Researchers have also experimented with modifying or extending the operations provided in tuple spaces and adapting the underlying semantics. One rather interesting proposal is to unify the concepts of tuples and tuple spaces by modelling everything as (unordered) sets – that is, tuple spaces are sets of tuples and tuples are sets of values, which may now also include tuples. This variant is known as Bauhaus Linda [Carriero *et al.* 1995].
- Perhaps most interestingly, recent implementations of tuple spaces have moved from tuples of typed data items to data objects (with attributes), turning the tuple space into an *object space*. This proposal is adopted, for example, in the influential system JavaSpaces, discussed in more detail below.

Implementation issues • Many of the implementations of tuple spaces adopt a centralized solution where the tuple space resource is managed by a single server. This has advantages in terms of simplicity, but such solutions are clearly not fault tolerant and also will not scale. Because of this, distributed solutions have been proposed.

Replication: Several systems have proposed the use of *replication* to overcome the problems identified above [Bakken and Schlichting 1995, Bessani *et al.* 2008, Xu and Liskov 1989].

The proposals from Bakken and Schlichting [1995] and Bessani *et al*. [2008] adopt a similar approach to replication, referred to as the *state machine approach* and discussed further in Chapter 18. This approach assumes that a tuple space behaves like a state machine, maintaining state and changing this state in response to events received from other replicas or from the environment. To ensure consistency the replicas (i) must start in the same state (an empty tuple space), (ii) must execute events in the same order and (iii) must react deterministically to each event. The key second property can be guaranteed by adopting a totally ordered multicast algorithm, as discussed in Section 6.2.2.

Xu and Liskov [1989] adopt a different approach, which optimizes the replication strategy by using the semantics of the particular tuple space operations. In this proposal, updates are carried out in the context of the current *view* (the agreed set of replicas) and tuples are also partitioned into distinct *tuple sets* based on their associated logical names (designated as the first field in the tuple). The system consists of a set of workers carrying out computations on the tuple space, and a set of tuple space replicas. A given physical node can contain any number of workers, replicas or indeed both; a given worker therefore may or may not have a local replica. Nodes are connected by a communications network that may lose, duplicate or delay messages and can deliver messages out of order. Network partitions can also occur.

A *write* operation is implemented by sending a multicast message over the unreliable communications channel to all members of the view. On receipt, members place this tuple into their replica and acknowledge receipt. The *write* request is repeated until all acknowledgements are received. For the correct operation of the protocol, replicas must detect and acknowledge duplicate requests, but not carry out the associated *write* operations.

The *read* operation consists of sending a multicast message to all replicas. Each replica seeks a match and returns this match to the requesting site. The first tuple returned is delivered as the result of the *read*. This may come from a local node, but given that many workers will not have a local replica, this is not guaranteed.

The *take* operation is more complex because of the need to agree on the tuple to be selected and to remove this agreed tuple from all copies. The algorithm proceeds in two phases. In phase 1, the tuple specification is sent to all replicas, and the replica attempts to acquired the lock on the associated tuple set to serialize *take* requests on the replicas (*write* and *read* operations are unaffected by the lock); if the lock cannot be acquired, the *take* request is refused. Each replica that succeeds in obtaining the lock responds with the *set* of matching tuples. This step is repeated until all replicas have accepted the request and responded. The initiating process can then select one tuple from the intersection of all the replies and return this as the result of the *take* request. If it is

Figure 6.21 Replication and the tuple space operations [Xu and Liskov 1989]

write
1. The requesting site multicasts the *write* request to all members of the view;
2. On receiving this request, members insert the tuple into their replica and acknowledge this action;
3. Step 1 is repeated until all acknowledgements are received.

read
1. The requesting site multicasts the *read* request to all members of the view;
2. On receiving this request, a member returns a matching tuple to the requestor;
3. The requestor returns the first matching tuple received as the result of the operation (ignoring others);
4. Step 1 is repeated until at least one response is received.

take

Phase 1: Selecting the tuple to be removed
1. The requesting site multicasts the *take* request to all members of the view;
2. On receiving this request, each replica acquires a lock on the associated tuple set and, if the lock cannot be acquired, the *take* request is rejected;
3. All accepting members reply with the set of all matching tuples;
4. Step 1 is repeated until all sites have accepted the request and responded with their set of tuples and the intersection is non-null;
5. A particular tuple is selected as the result of the operation (selected randomly from the intersection of all the replies);
6. If only a minority accept the request, this minority are asked to release their locks and phase 1 repeats.

Phase 2: Removing the selected tuple
1. The requesting site multicasts a *remove* request to all members of the view citing the tuple to be removed;
2. On receiving this request, members remove the tuple from their replica, send an acknowledgement and release the lock;
3. Step 1 is repeated until all acknowledgements are received.

not possible to obtain a majority of locks, the replicas are asked to release their locks and phase 1 repeats.

In phase 2, this tuple must be removed from all replicas. This is achieved by repeated multicasts to the replicas in the view until all have acknowledged deletion. As with *write* requests, it is necessary for replicas to detect repeat requests in phase 2 and to simply send another acknowledgement without carrying out another deletion (otherwise multiple tuples could erroneously be deleted at this stage).

The steps involved for each operation are summarized in Figure 6.21. Note that a separate algorithm is required to manage view changes if node failures occur or the network partitions (see Xu and Liskov [1989] for details).

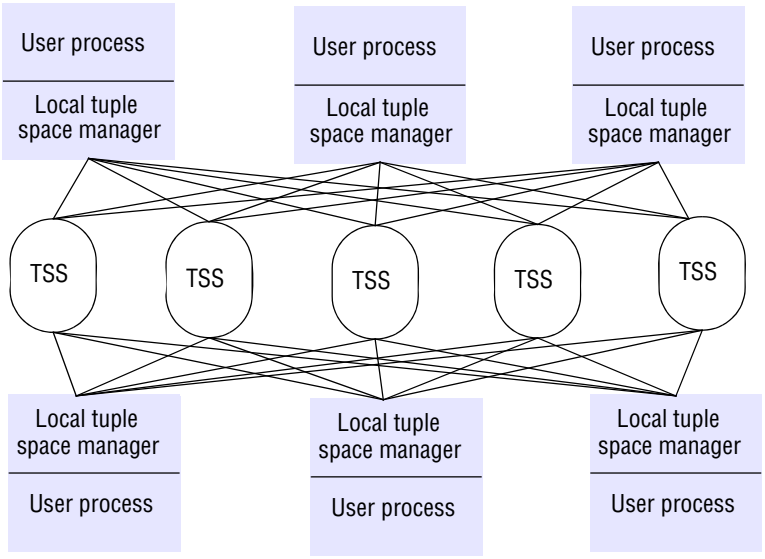
This algorithm is designed to minimize delay given the semantics of the three tuple space operations:

read operations only block until the first replica responds to the request.

take operations block until the end of phase 1, when the tuple to be deleted has been agreed.

write operations can return immediately.

Figure 6.22 Partitioning in the York Linda Kernel



This, though, introduces unacceptable levels of concurrency. For example, a *read* operation may access a tuple that should have been deleted in the second phase of a *take* operation. Therefore additional levels of concurrency control are required. In particular, Xu and Liskov [1989] introduce the following additional constraints:

- The operations of each worker must be executed at each replica in the same order as they were issued by the worker;.
- A *write* operation must not be executed at any replica until all previous *take* operations issued by the same worker have completed at all replicas in the worker's view.

A further example of using replication is provided in Chapter 19, where we present the L^{2}imbo approach, which uses replication to provide high availability in mobile environments [Davies *et al.* 1998].

Other approaches: A range of other approaches have been employed in the implementation of the tuple space abstraction, including partitioning of the tuple space over a number of nodes and mapping onto peer-to-peer overlays:

- The Linda Kernel developed at the University of York [Rowstron and Wood 1996] adopts an approach in which tuples are partitioned across a range of available tuple space servers (TSSs), as illustrated in Figure 6.22. There is no replication of tuples; that is, there is only one copy of each tuple. The motivation is to increase performance of the tuple space, especially for highly parallel computation.When a tuple is placed in tuple space, a hashing algorithm is used to select one of the tuple space servers to be used. The implementation of *read* or *take* is slightly more complex, as a tuple specification is provided that may specify types or values of the associated fields. The hashing algorithm uses this

Figure 6.23 The JavaSpaces API

Operation	*Effect*
Lease write(Entry e, Transaction txn, long lease)	Places an entry into a particular JavaSpace
Entry read(Entry tmpl, Transaction txn, long timeout)	Returns a copy of an entry matching a specified template
Entry readIfExists(Entry tmpl, Transaction txn, long timeout)	As above, but not blocking
Entry take(Entry tmpl, Transaction txn, long timeout)	Retrieves (and removes) an entry matching a specified template
Entry takeIfExists(Entry tmpl, Transaction txn, long timeout)	As above, but not blocking
EventRegistration notify(Entry tmpl, Transaction txn, RemoteEventListener listen, long lease, MarshalledObject handback)	Notifies a process if a tuple matching a specified template is written to a JavaSpace

specification to generate a set of possible servers that may contain matching tuples, and a linear search must then be employed until a matching tuple is discovered. Note that because there is only a single copy of a given tuple, the implementation of *take* is greatly simplified.

- Some implementations of tuple spaces have adopted peer-to-peer approaches in which all nodes cooperate to provide the tuple space service. This approach is particularly attractive given the intrinsic availability and scalability of peer-to-peer solutions. Examples of peer-to-peer implementations include PeerSpaces [Busi *et al.* 2003], which is developed using the JXTA peer-to-peer middleware [jxta.dev.java.net], LIME and TOTA (the latter two systems feature in Chapter 19).

Case study: JavaSpaces • JavaSpaces is a tool for tuple space communication developed by Sun [java.sun.com X, [java.sun.com VI]. More specifically, Sun provides the specification of a JavaSpaces service, and third-party developers are then free to offer implementations of JavaSpaces (significant implementations include GigaSpaces [www.gigaspaces.com] and Blitz [www.dancres.org]). The tool is strongly dependent on Jini (Sun's discovery service, discussed further in Section 19.2.1), as will become apparent below. The Jini Technology Starter Kit also includes an implementation of JavaSpaces, referred to as Outrigger.

The goals of the JavaSpaces technology are:

- to offer a platform that simplifies the design of distributed applications and services;
- to be simple and minimal in terms of the number and size of associated classes and to have a small footprint to allow the code to run on resource-limited devices (such as smart phones);
- to enable replicated implementations of the specification (although in practice most implementations are centralized).

Figure 6.24 Java class *AlarmTupleJS*

```
import net.jini.core.entry.*;

public class AlarmTupleJS implements Entry {
    public String alarmType;

    public AlarmTupleJS() {
    }

    public AlarmTupleJS(String alarmType) {
            this.alarmType = alarmType;
    }
}
```

Programming with JavaSpaces: JavaSpaces allows the programmer to create any number of instances of a *space*, where a space is a shared, persistent repository of *objects* (thus offering an object space in the terminology introduced above). More specifically, an item in a JavaSpace is referred to as an *entry*: a group of objects contained in a class that implements *net.jini.core.entry.Entry*. Note that with entries containing objects (rather than tuples), it is possible to associate arbitrary behaviour with entries, thus significantly increasing the expressive power of the approach.

The operations defined on JavaSpaces are summarized in Figure 6.23 (showing the full signatures of each of the operations) and can be described as follows:

- A process can place an entry into a JavaSpace instance with the *write* method. As with Jini, an entry can have an associated *lease* (see Section 5.4.3), which is the time for which access is granted to the associated objects. This can be forever (*Lease.FOREVER*) or can be a numerical value specified in milliseconds. After this period, the entry is destroyed. The *write* operation can also be used in the context of a *transaction*, as discussed below (a value of *null* indicates that this is not a transactional operation). The *write* operation returns a *Lease* value representing the lease granted by the JavaSpace (which may be less than the time requested).
- A process can access an entry in a JavaSpace with either the *read* or *take* operation; *read* returns a copy of a matching entry and *take* removes a matching entry from the JavaSpace (as in the general programming model presented above). The matching requirements are specified by a *template*, which is of type *entry*. Particular fields in the template may be set to specific values and others can be left unspecified. A match is then defined as an entry that is of the same class as the template (or a valid subclass) and where there is an exact match for the set of specified values. As with *write*, *read* and *take* can be carried out in the context of a specified transaction (discussed below). The two operations are also blocking; the final parameter specifies a timeout representing the maximum length of time that a particular process or thread will block, for example to deal with the failure of a process supplying a given entry. The *readIfExists* and *takeIfExists* operations

Figure 6.25 Java class *FireAlarmJS*

```
import net.jini.space.JavaSpace;

public class FireAlarmJS {

    public void raise() {
        try {
            JavaSpace space = SpaceAccessor.findSpace("AlarmSpace");
            AlarmTupleJS tuple = new AlarmTupleJS("Fire!");
            space.write(tuple, null, 60*60*1000);
        catch (Exception e) {
        }
    }
}
```

are equivalent to *read* and *take*, respectively, but these operations will return a matching entry if one exists; otherwise, they will return *null*.

- The *notify* operation uses Jini distributed event notification, mentioned in Section 6.3 to register an interest in a given event – in this case, the arrival of entries matching a given template. This registration is governed by a lease, that is, the length of time the registration should persist in the JavaSpace. Notification is via a specified *RemoteEventListener* interface. Once again, this operation can be carried out in the context of a specified transaction.

As mentioned throughout the discussion above, operations in JavaSpaces can take place in the context of a *transaction*, ensuring that either all or none of the operations will be executed. Transactions are distributed entities and can span multiple JavaSpaces and multiple participating processes. Discussion of the general concept of transactions is deferred until Chapter 16.

A simple example: We conclude this examination of JavaSpaces by presenting an example, the intelligent fire alarm example first introduced in Section 6.2.3 and revisited in Section 6.4.3. In this example, there is a need to disseminate an emergency message to all recipients when a fire event is detected.

We start by defining an entry object of type *AlarmTupleJS*, as shown in Figure 6.24. This is relatively straightforward and shows the creation of a new entry with one field, the *alarmType*. The associated fire alarm code is shown in Figure 6.25. The first step in raising an alarm is to gain access to an appropriate instance of a JavaSpace (called *"AlarmSpace"*), which we assume is already created. Most implementations of JavaSpaces provide utility functions for this and, for simplicity, this is what we show in this code, using a utility class *SpaceAccessor* and method *findSpace* as provided in GigaSpaces (for convenience, a copy of this class is provided on the companion web site for the book [www.cdk5.net]). An entry is then created as an instance of the previously defined *AlarmTupleJS*. This entry has only one field, a string called *alarmType*, and this

Figure 6.26 Java class *FireAlarmReceiverJS*

```
import net.jini.space.JavaSpace;

public class FireAlarmConsumerJS {

    public String await() {
        try {
            JavaSpace space = SpaceAccessor.findSpace();
            AlarmTupleJS template = new AlarmTupleJS("Fire!");
            AlarmTupleJS recvd = (AlarmTupleJS) space.read(template, null,
                                                    Long.MAX_VALUE);
            return recvd.alarmType;
        }
        catch (Exception e) {
            return null;
        }
    }
}
```

is set to *"Fire!"*". Finally, this entry is placed into the JavaSpace using the *write* method, where it will remain for one hour. This code can then be called using the following:

```
FireAlarmJS alarm = new FireAlarmJS();
alarm.raise();
```

The corresponding code for the consumer end is shown in Figure 6.26. Access to the appropriate JavaSpace is obtained in the same manner. Following this, a template is created, the single field is set to *"Fire!"*, and an associated *read* method is invoked. Note that by setting the field to *"Fire!"*, we ensure that only entries with this type *and* this value will be returned (leaving the field blank would make any entry of type *AlarmTupleJS* a valid match). This is called as follows in a consumer:

```
FireAlarmConsumerJS alarmCall = new FireAlarmConsumerJS();
String msg = alarmCall.await();
System.out.println("Alarm received: " + msg);
```

This simple example illustrates how easy it is to write multiparty applications using JavaSpaces that are both time- and space-uncoupled.

6.6 Summary

This chapter has examined indirect communication in detail, complementing the study of remote invocation paradigms in the previous chapter. We defined indirect communication in terms of communication through an intermediary, with a resultant uncoupling between producers and consumers of messages. This leads to interesting properties, particularly in terms of dealing with change and establishing fault-tolerant strategies.

We have considered five styles of indirect communication in this chapter:

- group communication;
- publish-subscribe systems;
- message queues;
- distributed shared memory;
- tuple spaces.

The discussion has emphasized their commonalities in terms of all supporting indirect communication through forms of intermediary including groups, channels or topics, queues, shared memory or tuple spaces. Content-based publish-subscribe systems communicate through the publish-subscribe system as a whole, with subscriptions effectively defining logical channels managed by content-based routing.

As well as focusing on the commonalities, it is instructive to consider the key differences between the various approaches. We start by reconsidering the level of space and time uncoupling, picking up on the discussion in Section 6.1. All the techniques considered in this chapter exhibit space uncoupling in that messages are directed to an intermediary and not to any specific recipient or recipients. The position with respect to time uncoupling is more subtle and dependent on the level of persistency in the paradigm. Message queues, distributed shared memory and tuple spaces all exhibit time uncoupling. The other paradigms may, depending on the implementation. For example, in group communication, it is possible in some implementations for a receiver to join a group at an arbitrary point in time and to be brought up-to-date with respect to previous message exchanges (this is an optional feature in JGroups, for example, selected by constructing an appropriate protocol stack). Many publish-subscribe systems do not support persistency of events and hence are not time-uncoupled, but there are exceptions. JMS, for example, does support persistent events, in keeping with its integration of publish-subscribe and message queues.

The next observation is that the initial three techniques (groups, publish-subscribe and message queues) offer a programming model that emphasizes *communication* (through messages or events), whereas distributed shared memory and tuple spaces offer a more *state-based abstraction*. This is a fundamental difference and one that has significant repercussions in terms of scalability; in general terms, the communication-based abstractions have the potential to scale to very large scale systems with appropriate routing infrastructure (although this is not the case for group communication because of the need to maintain group membership, as discussed in Section 6.2.2). In contrast, the two state-based approaches have limitations with respect to scaling. This stems from the need to maintain consistent views of the shared state, for example between multiple readers and writers of shared memory. The situation with tuple spaces is a bit more subtle given the immutable nature of tuples. The key problem rests with implementing the destructive read operation, *take*, in a large-scale system; it is an interesting observation that without this operation, tuple spaces look very much like publish-subscribe systems (and hence are potentially highly scalable).

Most of the above systems also offer *one-to-many* styles of communication, that is, multicast in terms of the communication-based services and global access to shared values in the state-based abstractions. The exceptions are message queuing, which is fundamentally *point-to-point* (and hence often offered in combination with publish-

Figure 6.27 Summary of indirect communication styles

	Groups	*Publish-subscribe systems*	*Message queues*	*DSM*	*Tuple spaces*
Space-uncoupled	Yes	Yes	Yes	Yes	Yes
Time-uncoupled	Possible	Possible	Yes	Yes	Yes
Style of service	Communication-based	Communication-based	Communication-based	State-based	State-based
Communication pattern	1-to-many	1-to-many	1-to-1	1-to-many	1-1 or 1-to-many
Main intent	Reliable distributed computing	Information dissemination or EAI; mobile and ubiquitous systems	Information dissemination or EAI; commercial transaction processing	Parallel and distributed computation	Parallel and distributed computation; mobile and ubiquitous systems
Scalability	Limited	Possible	Possible	Limited	Limited
Associative	No	Content-based publish-subscribe only	No	No	Yes

subscribe systems in commercial middleware), tuple spaces, which can be either one-to-many or point-to-point depending on whether receiving processes use the *read* or *take* operations, respectively.

There are also differences in *intent* in the various systems. Group communication is mainly designed to support reliable distributed systems, and hence the emphasis is on providing algorithmic support for reliability and ordering of message delivery. Interestingly, the algorithms to ensure reliability and ordering (especially the latter) can have a significant negative effect on scalability for similar reasons to maintaining consistent views of shared state. Publish-subscribe systems have largely been targeted at information dissemination (for example, in financial systems) and for Enterprise Application Integration. Finally, the shared memory approaches have generally been applied in parallel and distributed processing, including in the Grid community (although tuple spaces have been used effectively across a variety of application domains). Both publish-subscribe systems and tuple space communication have found favour in mobile and ubiquitous computing due to their support for volatile environments (as discussed in Chapter 19).

One other key issue associated with the five schemes is that both content-based publish-subscribe and tuple spaces offer a form of *associative addressing* based on content, allowing pattern matching between subscriptions and events or templates against tuples, respectively. The other approaches do not.

This discussion is summarized in Figure 6.27.

We have not considered issues related to quality of service in this analysis. Many message queue systems do offer intrinsic support for reliability in the form of transactions. More generally, however, quality of service remains a key challenge for indirect communication paradigms. Indeed, space and time uncoupling by their very nature make it difficult to reason about end-to-end properties of the system, such as real-time behaviour or security, and hence this is an important area for further research.

EXERCISES

6.1 Construct an argument as to why indirect communication may be appropriate in volatile environments. To what extent can this be traced to time uncoupling, space uncoupling or indeed a combination of both? *page 230*

6.2 Section 6.1 states that message passing is both time- and space-coupled – that is, messages are both directed towards a particular entity and require the receiver to be present at the time of the message send. Consider the case, though, where messages are directed towards a name rather than an address and this name is resolved using DNS. Does such a system exhibit the same level of indirection? *page 231, Section 13.2.3*

6.3 Section 6.1 refers to systems that are space-coupled but time- uncoupled – that is, messages are directed towards a given receiver (or receivers), but that receiver can have a lifetime independent from the sender's. Can you construct a communication paradigm with these properties? For example, does email fall into this category? *page 231*

6.4 As a second example, consider the communication paradigm referred to as queued RPC, as introduced in Rover [Joseph *et al*. 1997]. Rover is a toolkit to support distributed systems programming in mobile environments where participants in communication may become disconnected for periods of time. The system offers the RPC paradigm and hence calls are directed towards a given server (clearly space-coupled). The calls, though, are routed through an intermediary, a queue at the *sending* side, and are maintained in the queue until the receiver is available. To what extent is this time-uncoupled? Hint: consider the almost philosophical question of whether a recipient that is temporarily unavailable exists at that point in time. *page 231, Chapter 19*

6.5 If a communication paradigm is asynchronous, is it also time-uncoupled? Explain your answer with examples as appropriate. *page 232*

6.6 In the context of a group communication service, provide example message exchanges that illustrate the difference between causal and total ordering. *page 236*

6.7 Consider the *FireAlarm* example as written using JGroups (Section 6.2.3). Suppose this was generalized to support a variety of alarm types, such as fire, flood, intrusion and so on. What are the requirements of this application in terms of reliability and ordering? *page 230, page 240*

6.8 Suggest a design for a notification mailbox service that is intended to store notifications on behalf of multiple subscribers, allowing subscribers to specify when they require notifications to be delivered. Explain how subscribers that are not always active can make use of the service you describe. How will the service deal with subscribers that crash while they have delivery turned on? *page 245*

6.9 In publish-subscribe systems, explain how channel-based approaches can trivially be implemented using a group communication service? Why is this a less optimal strategy for implementing a content-based approach? *page 245*

6.10 Using the filtering-based routing algorithm in Figure 6.11 as a starting point, develop an alternative algorithm that illustrates how the use of advertisements can result in significant optimization in terms of message traffic generated. *page 251*

6.11 Construct a step-by-step guide explaining the operation of the alternative rendezvous-based routing algorithm shown in Figure 6.12. *page 252*

6.12 Building on your answer to Exercise 6.11, discuss two possible implementations of *EN(e)* and *SN(s)*. Why must the intersection of *EN(e)* and *SN(s)* be non-null for a given *e* that matches *s* (the intersection rule)? Does this apply in your possible implementations? *page 252*

6.13 Explain how the loose coupling inherent in message queues can aid with Enterprise Application Integration. As in Exercise 6.1, consider to what extent this can be traced to time uncoupling, space uncoupling or a combination of both. *page 254*

6.14 Consider the version of the *FireAlarm* program written in JMS (Section 6.4.3). How would you extend the consumer to receive alarms only from a given location? *page 261*

6.15 Explain in which respects DSM is suitable or unsuitable for client-server systems. *page 262*

6.16 Discuss whether message passing or DSM is preferable for fault-tolerant applications. *page 262*

6.17 Assuming a DSM system is implemented in middleware without any hardware support and in a platform-neutral manner, how would you deal with the problem of differing data representations on heterogeneous computers? Does your solution extend to pointers? *page 262*

6.18 How would you implement the equivalent of a remote procedure call using a tuple space? What are the advantages and disadvantages of implementing a remote procedure call–style interaction in this way? *page 265*

6.19 How would you implement a semaphore using a tuple space? *page 265*

6.20 Implement a replicated tuple space using the algorithm of Xu and Liskov [1989]. Explain how this algorithm uses the semantics of tuple space operations to optimize the replication strategy. *page 269*

REFERENCES

Online references

This reference list is available on the Web at www.cdk5.net/refs. It provides clickable links for those documents that exist only on the Web. In this printed list items identified by an underlined tag, for example, www.omg.org, point to an index page leading to the document; direct links can be found in the online reference list.

The references to RFCs refer to the series of Internet standards and specifications called 'requests for comments' that are available from the Internet Engineering Task Force at www.ietf.org/rfc/ and several other well-known sites.

The online reference list may also be used as an aid to searching for web copies of other documents by searching for authors or titles with Google or Citeseer at citeseer.ist.psu.edu.

Online material written or edited by the authors to supplement the book is referenced in the book by a www.cdk5.net tag but is not included in the reference list. For example, www.cdk5.net/ipc refers to the additional material about interprocess communication on our web pages.

.

Abadi and Gordon 1999 — Abadi, M. and Gordon, A.D. (1999). A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, Vol. 148, No. 1, pp. 1–70.

Abadi *et al.* 1998 — Abadi, M., Birrell, A.D., Stata, R. and Wobber, E.P. (1998). Secure Wweb tunneling. In *Proceedings of the 7th International World Wide Web Conference*, pp. 531–9. Elsevier, in *Computer Networks and ISDN Systems*, Volume 30, Nos 1–7.

Abrossimov *et al.* 1989 — Abrossimov, V., Rozier, M. and Shapiro, M. (1989). Generic virtual memory management for operating system kernels. *Proceedings of 12th ACM Symposium on Operating System Principles*, December, pp. 123–36.

Accetta *et al.* 1986 — Accetta, M., Baron, R., Golub, D., Rashid, R., Tevanian, A. and Young, M. (1986). Mach: A new kernel foundation for UNIX development. In *Proceedings of the Summer 1986 USENIX Conference*, pp. 93–112.

Adjie-Winoto *et al.* 1999 — Adjie-Winoto, W., Schwartz, E., Balakrishnan, H. and Lilley, J. (1999). The design and implementation of an intentional naming system. In *Proceedings of the 17th ACM Symposium on Operating System Principles*, published as *Operating Systems Review*, Vol. 34, No. 5, pp. 186–201.

Agrawal *et al.* 1987 — Agrawal, D., Bernstein, A., Gupta, P. and Sengupta, S. (1987). Distributed optimistic concurrency control with reduced rollback. *Distributed Computing*, Vol. 2, No 1, pp. 45–59.

Ahamad *et al.* 1992 — Ahamad, M., Bazzi, R., John, R., Kohli, P. and Neiger, G. (1992). *The Power of Processor Consistency*. Technical report GIT-CC-92/34, Georgia Institute of Technology, Atlanta, GA.

Al-Muhtadi *et al*. 2002 — Al-Muhtadi, J., Campbell, R., Kapadia, A., Mickunas, D. and Yi, S. (2002). Routing through the mist: Privacy preserving communication in ubiquitous computing environments. In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS'02)*, Vienna, Austria, July, pp. 74–83.

Albanna *et al.* 2001 — Albanna, Z., Almeroth, K., Meyer, D. and Schipper, M. (2001). *IANA Guidelines for IPv4 Multicast Address Assignments*. Internet RFC 3171.

Alonso *et al*. 2004 — Alonso, G., Casata, C., Kuno, H. and Machiraju, V. (2004). *Web Services, Concepts, Architectures and Application*s. Berlin, Heidelberg: Springer-Verlag.

Anderson 1993 — Anderson, D.P. (1993). Metascheduling for continuous media. *ACM Transactions on Computer Systems*, Vol. 11, No. 3, pp. 226–52.

Anderson 1996 — Anderson, R. J. (1996). The Eternity Service. In *Proceedings of Pragocrypt '96*, pp. 242–52.

Anderson 2008 — Anderson, R.J. (2008). *Security Engineering*, 2nd edn. John Wiley & Sons.

Anderson *et al.* 1990 — Anderson, D.P., Herrtwich, R.G. and Schaefer, C. (1990). *SRP – A Resource Reservation Protocol for Guaranteed-Performance Communication in the Internet*. Technical report 90-006, International Computer Science Institute, Berkeley, CA.

Anderson *et al.* 1991 — Anderson, T., Bershad, B., Lazowska, E. and Levy, H. (1991). Scheduler activations: Efficient kernel support for the user-level management of parallelism. In *Proceedings of the 13th ACM Symposium on Operating System Principles*, pp. 95–109.

Anderson *et al.* 1995 — Anderson, T., Culler, D., Patterson, D. and the NOW team. (1995). A case for NOW (Networks Of Workstations). *IEEE Micro*, Vol. 15, No. 1, pp. 54–64.

Anderson *et al.* 1996 — Anderson, T.E., Dahlin, M.D., Neefe, J.M., Patterson, D.A., Roselli, D.S. and Wang, R.Y. (1996). Serverless Network File Systems. *ACM Trans. on Computer Systems*, Vol. 14, No. 1, pp. 41–79.

Anderson *et al.* 2002	Anderson, D.P., Cobb, J., Korpela, E., Lebofsky, M. and Werthimer, D. (2002). SETI@home: An experiment in public-resource computing. *Communications of the ACM*, Vol. 45, No. 11, pp. 56–61.
Anderson *et al*. 2004	Anderson, R., Chan, H. and Perrig, A. (2004). Key infection: Smart trust for smart dust. In *Proceedings of the IEEE 12th International Conference on Network Protocols* (ICNP 2004), Berlin, Germany, October, pp. 206–215.
ANSA 1989	ANSA (1989). *The Advanced Network Systems Architecture (ANSA) Reference Manual*. Castle Hill, Cambridge, England: Architecture Project Management.
ANSI 1985	American National Standards Institute (1985). *American National Standard for Financial Institution Key Management*. Standard X9.17 (revised).
Armand *et al.* 1989	Armand, F., Gien, M., Herrman, F. and Rozier, M. (1989). Distributing UNIX brings it back to its original virtues. In *Proc. Workshop on Experiences with Building Distributed and Multiprocessor Systems*, October, pp. 153–174.
Arnold *et al.* 1999	Arnold, K., O'Sullivan, B., Scheifler, R.W., Waldo, J. and Wollrath, A. (1999). *The Jini Specification*. Reading, MA: Addison-Wesley.
associates.amazon.com	Amazon Web Service FAQs.
Attiya and Welch 1998	Attiya, H. and Welch, J. (1998). *Distributed Computing – Fundamentals, Simulations and Advanced Topics*. Maidenhead, England: McGraw-Hill.
aws.amazon.com	Amazon Web Services. *Home page*.
Babaoglu *et al.* 1998	Babaoglu, O., Davoli, R., Montresor, A. and Segala, R. (1998). System support for partition-aware network applications. In *Proceedings of the 18th International Conference on Distributed Computing Systems* (ICDCS '98), pp. 184–191.
Bacon 2002	Bacon, J. (2002). *Concurrent Systems*, 3rd edn. Harlow, England: Addison-Wesley.
Baker 1997	Baker, S. (1997). *CORBA Distributed Objects Using Orbix.* Harlow, England: Addison-Wesley.
Bakken and Schlichting 1995	Bakken, D.E. and Schlichting, R.D. (1995). Supporting fault-tolerant parallel programming in Linda. *IEEE Transactions Parallel and Distributed Systems*, Vol. 6, No. 3, pp. 287–302.
Balakrishnan *et al.* 1995	Balakrishnan, H., Seshan, S. and Katz, R.H. (1995). Improving reliable transport and hand-off performance in cellular wireless networks. In *Proceedings of the ACM Mobile Computing and Networking Conference*, pp. 2–11.

Balakrishnan *et al.* 1996 — Balakrishnan, H., Padmanabhan, V., Seshan, S. and Katz, R. (1996). A comparison of mechanisms for improving TCP performance over wireless links. In *Proceedings of the ACM SIGCOMM '96 Conference*, pp. 256–69.

Balan *et al*. 2003 — Balan, R.K., Satyanarayanan, M., Park, S., Okoshi, T. (2003). Tactics-based remote execution for mobile computing. In *Proceedings of the First USENIX International Conference on Mobile Systems, Applications, and Services (MobiSys 2003)*, San Francisco, CA, May, pp. 273–286.

Balazinska *et al*. 2002 — Balazinska, M., Balakrishnan, H. and Karger, D. (2002). INS/Twine: A scalable peer-to-peer architecture for intentional resource discovery. In *Proceedings of the International Conference on Pervasive Computing*, Zurich, Switzerland, August, pp. 195–210.

Baldoni *et al*. 2005 — Baldoni, R. Beraldi, R., Cugola, G., Migliavacca, M. and Querzoni, L. (2005). Structure-less content-based routing in mobile ad hoc networks. In *Proceedings of the International Conference on Pervasive Services*, pp. 37–46.

Baldoni and Virgillito 2005 — Baldoni, R. and Virgillito, A. (2005). *Distributed event routing in publish/subscribe communication systems: A survey*. Technical Report 15-05, Dipartimento di Informatica e Sistemistica, Universita di Roma "La Sapienza".

Baldoni *et al*. 2007 — Baldoni, R., Beraldi, R., Quema, V., Querzoni, L. and Tucci-Piergiovanni, S. (2007). TERA: Topic-based event routing for peer-to-peer architectures. In *Proceedings of the 2007 Inaugural International Conference on Distributed Event-Based Systems*. Toronto, Ontario, Canada, June, pp. 2–13.

Balfanz *et al*. 2002 — Balfanz, D., Smetters, D.K., Stewart, P. and Wong, H.C. (2002). Talking to strangers: Authentication in ad-hoc wireless networks. In *Proceedings of the Network and Distributed System Security Symposium*, San Diego, CA, February.

Banerjea and Mah 1991 — Banerjea, A. and Mah, B.A. (1991). The real-time channel administration protocol. *Second International Workshop on Network and Operating System Support for Digital Audio and Video*, Heidelberg, Germany.

Baran 1964 — Baran, P. (1964). *On Distributed Communications*. Research Memorandum RM-3420-PR, Rand Corporation.

Barborak *et al.* 1993 — Barborak, M., Malek, M. and Dahbura, A. (1993). The consensus problem in fault-tolerant computing. *ACM Computing Surveys*, Vol. 25, No. 2, pp. 171–220.

Barghouti and Kaiser 1991 — Barghouti, N.S. and Kaiser, G.E. (1991). Concurrency control in advanced database applications. *ACM Computing Surveys*, Vol. 23, No. 3, pp. 269–318.

Barham *et al*. 2003a — Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I. and Warfield, A. (2003). Xen and the art of virtualization. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, Bolton Landing, NY, October, pp. 164–177.

Barham *et al*. 2003b — Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A.,Kotsovinos, E., Madhavapeddy, A.V.S., Neugebauer, R., Pratt, I. and Warfield, A. (2003). *Xen 2002*. Technical Report UCAM-CL-TR-553, Computing Laboratory, University of Cambridge.

Barr and Asanovic 2003 — Barr, K. and Asanovic, K. (2003). Energy aware lossless data compression. *Proceedings of the First USENIX International Conference on Mobile Systems, Applications, and Services (MobiSys 2003)*, San Francisco, CA, May, pp. 231–244.

Barton *et al*. 2002 — Barton, J., Kindberg, T. and Sadalgi, S. (2002). Physical registration: Configuring electronic directories using handheld devices. *IEEE Wireless Communications*, Vol. 9, No. 1, pp. 30–38.

Baset and Schulzrinne 2006 — Baset, S.A. and Schulzrinne, H.G. (2006). An analysis of the Skype peer-to-peer Internet telephony protocol. In *Proceedings of the 25th IEEE International Conference on Computer Communications (INFOCOM'06)*, pp. 1–11.

Bates *et al.* 1996 — Bates, J., Bacon, J., Moody, K. and Spiteri, M. (1996). Using events for the scalable federation of heterogeneous components. *European SIGOPS Workshop*.

Baude *et al*. 2009 — Baude, F., Caromel, D., Dalmasso, C., Danelutto, M., Getov, V., Henrio, L. and Pérez, C. (2009). GCM: A grid extension for Fractal autonomous distributed components. *Annals of Telecommunications*, Springer, Vol. 64, No. 1, pp. 5–24.

Bell and LaPadula 1975 — Bell, D.E. and LaPadula, L.J. (1975). *Computer Security Model: Unified Exposition and Multics Interpretation*. Mitre Corporation.

Bellman 1957 — Bellman, R.E. (1957). *Dynamic Programming*. Princeton, NJ: Princeton University Press.

Bellovin and Merritt 1990 — Bellovin, S.M. and Merritt, M. (1990). Limitations of the Kerberos authentication system. *ACM Computer Communications Review*, Vol. 20, No. 5, pp. 119–32.

Bellwood *et al.* 2003 — Bellwood, T., Clément, L. and von Riegen, C. (eds.) (2003). *UDDI Version 3.0.1*. Oasis Corporation.

Beresford and Stajano 2003 — Beresford, A. and Stajano, F. (2003). Location privacy in pervasive computing. *IEEE Pervasive Computing*, Vol. 2, No. 1, pp. 46–55.

Berners-Lee 1991	Berners-Lee, T. (1991). World Wide Web Seminar.
Berners-Lee 1999	Berners-Lee, T. (1999). *Weaving the Web*. New York: HarperCollins.
Berners-Lee *et al.* 2005	Berners Lee, T., Fielding, R. and Masinter, L. (2005). Uniform Resource Identifiers (URI): Generic syntax. Internet RFC 3986.
Bernstein *et al.* 1980	Bernstein, P.A., Shipman, D.W. and Rothnie, J.B. (1980). Concurrency control in a system for distributed databases (SDD-1). *ACM Transactions on Database Systems*, Vol. 5, No. 1, pp. 18–51.
Bernstein *et al.* 1987	Bernstein, P., Hadzilacos, V. and Goodman, N. (1987). *Concurrency Control and Recovery in Database Systems*. Reading, MA: Addison-Wesley. Text available online.
Bershad *et al.* 1990	Bershad, B., Anderson, T., Lazowska, E. and Levy, H. (1990). Lightweight remote procedure call. *ACM Transactions on Computer Systems*, Vol. 8, No. 1, pp. 37–55.
Bershad *et al.* 1991	Bershad, B., Anderson, T., Lazowska, E. and Levy, H. (1991). User-level interprocess communication for shared memory multiprocessors. *ACM Transactions on Computer Systems*, Vol. 9, No. 2, pp. 175–198.
Bershad *et al.* 1995	Bershad, B., Savage, S., Pardyak, P., Sirer, E., Fiuczynski, M., Becker, D., Chambers, C. and Eggers, S. (1995). Safety and performance in the SPIN operating system. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles*, pp. 267–84.
Bessani *et al.* 2008	Bessani, A. N., Alchieri, E. P., Correia, M. and Fraga, J. S. (2008). DepSpace: A Byzantine fault-tolerant coordination service. In *Proceedings of the 3rd ACM Sigops/Eurosys European Conference on Computer Systems*. Glasgow, Scotland, April, pp.163-176.
Bhagwan *et al.* 2003	Bhagwan, R., Savage, S. and Voelker, G. (2003). Understanding availability. In *Proc. 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, Berkeley, CA, February.
Bhatti and Friedrich 1999	Bhatti, N. and Friedrich, R. (1999). *Web Server Support for Tiered Services*. Hewlett-Packard Corporation Technical Report HPL-1999-160.
Birman 1993	Birman, K.P. (1993). The process group approach to reliable distributed computing. *Comms. ACM*, Vol. 36, No. 12, pp. 36–53.
Birman 2004	Birman, K.P. (2004). Like it or not, web services are distributed Objects! *Comms. ACM*. Vol. 47, No. 12, pp. 60–62. .
Birman 2005	Birman, K.P. (2005). *Reliable Distributed Systems: Technologies, Web Services and Applications.* Springer-Verlag.
Birman and Joseph 1987a	Birman, K.P. and Joseph, T.A. (1987). Reliable communication in the presence of failures. *ACM Transactions on Computer Systems*, Vol. 5, No. 1, pp. 47–76.

Birman and Joseph 1987b — Birman, K. and Joseph, T. (1987). Exploiting virtual synchrony in distributed systems. In *Proceedings of the 11th ACM Symposium on Operating Systems Principles*, pp. 123–38.

Birman *et al.* 1991 — Birman, K.P., Schiper, A. and Stephenson, P. (1991). Lightweight causal and atomic group multicast. *ACM Transactions on Computer Systems*, Vol. 9, No. 3, pp. 272–314.

Birrell and Needham 1980 — Birrell, A.D. and Needham, R.M. (1980). A universal file server. *IEEE Transactions Software Engineering*, Vol. SE-6, No. 5, pp. 450–3.

Birrell and Nelson 1984 — Birrell, A.D. and Nelson, B.J. (1984). Implementing remote procedure calls. *ACM Transactions on Computer Systems*, Vol. 2, No. 1, pp. 39–59.

Birrell *et al.* 1982 — Birrell, A.D., Levin, R., Needham, R.M. and Schroeder, M.D. (1982). Grapevine:An exercise in distributed computing. *Comms. ACM*, Vol. 25, No. 4, pp. 260–73.

Birrell *et al.* 1995 — Birrell, A., Nelson, G. and Owicki, S. (1993). Network objects. In *Proceedings of the 14th ACM Symposium on Operating Systems Principles*, pp. 217–30.

Bisiani and Forin 1988 — Bisiani, R. and Forin, A. (1988). Multilanguage parallel programming of heterogeneous machines. *IEEE Transactions Computers*, Vol. 37, No. 8, pp. 930–45.

Black 1990 — Black, D. (1990). Scheduling support for concurrency and parallelism in the Mach operating system. *IEEE Computer*, Vol. 23, No. 5, pp. 35–43.

Blakley 1999 — Blakley, R. (1999). *CORBA Security – An Introduction to Safe Computing with Objects*. Reading, MA: Addison-Wesley.

Bloch 2006 — Bloch, J. (2006). How to design a good API and why it matters. In *Companion to the 21st ACM SIGPLAN Symposium on Object-Oriented Programming Systems, Languages, and Applications*. (OOPSLA'06), Portland, Oregon, pp. 506–507.

Bolosky *et al.* 1996 — Bolosky, W., Barrera, J., Draves, R., Fitzgerald, R., Gibson, G., Jones, M., Levi, S., Myhrvold, N. and Rashid, R. (1996). The Tiger video fileserver. *6th NOSSDAV Conference*, Zushi, Japan, April.

Bolosky *et al.* 1997 — Bolosky, W., Fitzgerald, R. and Douceur, J. (1997). Distributed schedule management in the Tiger video fileserver. In *Proc. of the 16th ACM Symposium on Operating System Principles*, St Malo, France, October, pp. 212–23.

Bolosky *et al.* 2000 — Bolosky, W.J., Douceur, J.R., Ely, D. and Theimer, M. (2000). Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs. In *Proceedings of the International Conference on Measurement and Modeling of Computer Systems*, pp. 34–43.

Bonnaire *et al.* 1995	Bonnaire, X., Baggio, A. and Prun, D. (1995). Intrusion free monitoring: An observation engine for message server based applications. In *Proceedings of the 10th International Symposium on Computer and Information Sciences* (ISCIS X), pp. 541–48.
Booch *et al.* 2005	Booch, G., Rumbaugh, J. and Jacobson, I. (2005). *The Unified Modeling Language User Guide,* 2nd edn. Reading MA: Addison-Wesley.
Borisov *et al*. 2001	Borisov, N., Goldberg, I. and Wagner, D. (2001). Intercepting mobile communications: The insecurity of 802.11. In *Proceedings of MOBICOM 2001, pp. 180–9*.
Bowman *et al.* 1990	Bowman, M., Peterson, L. and Yeatts, A. (1990). Univers: An attribute-based name server. *Software–Practice and Experience*, Vol. 20, No. 4, pp. 403–24.
Box 1998	Box, D. (1998). *Essential COM*. Reading, MA: Addison-Wesley.
Box and Curbera 2004	Box, D. and Curbera, F. (2004). *Web Services Addressing (WS-Addressing*). BEA Systems, IBM and Microsoft, August.
boxee.tv	Boxee. *Home page*.
Boykin *et al.* 1993	Boykin, J., Kirschen, D., Langerman, A. and LoVerso, S. (1993). *Programming under Mach*. Reading, MA: Addison-Wesley.
Bray and Sturman 2002	Bray, J. and Sturman. C.F. (2002). *Bluetooth: Connect Without Cables*, 2nd edn. Upper Saddle River, NJ: Prentice-Hall.
Brin and Page 1998	Brin, S. and Page, L. (1998). The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, Vol. 30, Issue 1-7, pp. 107-117.
Bruneton *et al*. 2006	Bruneton, E., Coupaye, T., LeClercq, M., Quema, V. and Stefani, J.B. (2006). The Fractal component model and its support in Java. *Software – Practice and Experience*, Vol. 36, Nos.11–21, pp. 1257–1284.
Buford 1994	Buford, J.K. (1994). *Multimedia Systems*. Addison-Wesley.
Burns and Wellings 1998	Burns, A. and Wellings, A. (1998). *Concurrency in Ada*. England: Cambridge University Press.
Burrows *et al.* 1989	Burrows, M., Abadi, M. and Needham, R. (1989). *A logic of authentication*. Technical Report 39. Palo Alto, CA: Digital Equipment Corporation Systems Research Center.
Burrows *et al.* 1990	Burrows, M., Abadi, M. and Needham, R. (1990). A logic of authentication. *ACM Transactions on Computer Systems*, Vol. 8, No. 1, pp. 18–36.
Burrows 2006	Burrows, M. (2006). The Chubby lock service for loosely-coupled distributed systems. In *Proc. of the 7th Symposium on Operating Systems Design and Implementation*, Seattle, WA, pp. 335–350.
Bush 1945	Bush, V. (1945). As we may think. *The Atlantic Monthly*, July.

Bushmann *et al*. 2007 — Bushmann, F., Henney, K. and Schmidt, D.C. (2007). *Pattern-Oriented Software Architecture: A Pattern for Distributed Computing*, New York: John Wiley & Sons.

Busi *et al.* 2003 — Busi, N., Manfredini, C., Montresor, A. and Zavattaro, G. (2003). PeerSpaces: Data-driven coordination in peer-to-peer networks. In *Proceedings of the 2003 ACM Symposium on Applied Computing*, Melbourne, Florida, March, pp. 380–386.

Callaghan 1996a — Callaghan, B. (1996). *WebNFS Client Specification*. Internet RFC 2054.

Callaghan 1996b — Callaghan, B. (1996). *WebNFS Server Specification*. Internet RFC 2055.

Callaghan 1999 — Callaghan, B. (1999). *NFS Illustrated*. Reading, MA: Addison-Wesley.

Callaghan *et al.* 1995 — Callaghan, B., Pawlowski, B. and Staubach, P. (1995). *NFS Version 3 Protocol Specification.* Internet RFC 1813, Sun Microsystems.

Campbell 1997 — Campbell, R. (1997). *Managing AFS: The Andrew File System*. Upper Saddle River, NJ: Prentice-Hall.

Campbell *et al.* 1993 — Campbell, R., Islam, N., Raila, D. and Madany, P. (1993). Designing and implementing Choices: An object-oriented system in C++. *Comms. ACM*, Vol. 36, No. 9, pp. 117–26.

Canetti and Rabin 1993 — Canetti, R. and Rabin, T. (1993). Fast asynchronous Byzantine agreement with optimal resilience. In *Proceedings of the 25th ACM Symposium on Theory of Computing*, pp. 42–51.

Cao and Singh 2005 — Cao, F. and Singh, J. P. (2005). MEDYM: match-early with dynamic multicast for content-based publish-subscribe networks. In *Proceedings of the ACM/IFIP/USENIX 2005 International Conference on Middleware*, Grenoble, France, November, pp. 292–313.

Carriero *et al.* 1995 — Carriero, N., Gelernter, D. and Zuck, L. (1995). Bauhaus Linda. In *LNCS 924: Object-based models and languages for concurrent systems*. Berlin, Heidelberg: Springer-Verlag. pp. 66–76.

Carter *et al.* 1991 — Carter, J.B., Bennett, J.K. and Zwaenepoel, W. (1991). Implementation and performance of Munin. *In Proceedings of the 13th ACM Symposium on Operating System Principles*, pp. 152–64.

Carter *et al.* 1998 — Carter, J., Ranganathan, A. and Susarla, S. (1998). Khazana, an infrastructure for building distributed services. In *Proceedings of ICDCS '98*. Amsterdam, The Netherland, pp. 562–71.

Carzaniga *et al.* 2001 — Carzaniga, A., Rosenblum, D. S. and Wolf, A. L. (2001). Design and evaluation of a wide-area event notification service. *ACM Trans. on Compuer Systems*, Vol. 19, No. 3, pp. 332–383.

Castro and Liskov 2000 — Castro, M. and Liskov, B. (2000). Proactive recovery in a Byzantine-fault-tolerant system. *Proceedings of the Fourth Symposium on Operating Systems Design and Implementation (OSDI '00)*, San Diego, CA, October, p. 19.

Castro *et al.* 2002a — Castro, M., Druschel, P., Hu, Y.C. and Rowstron, A. (2002). Topology-aware routing in structured peer-to-peer overlay networks. *Technical Report MSR-TR-2002-82*, Microsoft Research, 2002.

Castro *et al.* 2002b — Castro, M., Druschel, P., Kermarrec, and Rowstron, A. (2002). SCRIBE: A large-scale and decentralised application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communication (JSAC)*, Vol. 20, No, 8, pp. 1489–99.

Castro *et al.* 2003 — Castro, M., Costa, M. and Rowstron, A. (2003). *Performance and dependability of structured peer-to-peer overlays*. Technical Report MSR-TR-2003-94, Microsoft Research, 2003.

CCITT 1988a — CCITT (1988). *Recommendation X.500*: *The Directory – Overview of Concepts, Models and Service*. International Telecommunications Union, Place des Nations, 1211 Geneva, Switzerland.

CCITT 1988b — CCITT (1988). *Recommendation X.509: The Directory – Authentication Framework.* International Telecommunications Union, Place des Nations, 1211 Geneva, Switzerland.

CCITT 1990 — CCITT (1990). *Recommendation I.150*: *B-ISDN ATM Functional Characteristics*. International Telecommunications Union, Place des Nations, 1211 Geneva, Switzerland.

Ceri and Owicki 1982 — Ceri, S. and Owicki, S. (1982). On the use of optimistic methods for concurrency control in distributed databases. In *Proceedings of the 6th Berkeley Workshop on Distributed Data Management and Computer Networks*, Berkeley, CA, pp. 117–30.

Ceri and Pelagatti 1985 — Ceri, S. and Pelagatti, G. (1985). *Distributed Databases – Principles and Systems*. Maidenhead, England: McGraw-Hill.

Chalmers *et al*. 2004 — Chalmers, D., Dulay, N. and Sloman, M. (2004). Meta data to support context aware mobile applications. In *Proceedings of the IEEE Intl. Conference on Mobile Data Management (MDM 2004)*, Berkeley, CA, January, pp. 199–210.

Chandra and Toueg 1996 — Chandra, T. and Toueg, S. (1996). Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*. Vol 43, No. 2, pp. 225–67.

Chandra *et al*. 2007 — Chandra, T. D., Griesemer, R. and Redstone, J. (2007). Paxos made live: An engineering perspective. In *Proc. of the Twenty-Sixth Annual ACM Symposium on Principles of Distributed Computing, (*PODC'07), Portland, Oregon, pp. 398–407.

Chandy and Lamport 1985 — Chandy, K. and Lamport, L. (1985). Distributed snapshots: Determining global states of distributed systems. *ACM Transactions on Computer Systems*, Vol. 3, No. 1, pp. 63–75.

Chang and Maxemchuk 1984 — Chang, J. and Maxemchuk, N. (1984). Reliable broadcast protocols. *ACM Transactions on Computer Systems*, Vol. 2, No. 3, pp. 251–75.

Chang and Roberts 1979 — Chang, E.G. and Roberts, R. (1979). An improved algorithm for decentralized extrema-finding in circular configurations of processors. *Comms. ACM*, Vol. 22, No. 5, pp. 281–3.

Chang *et al*. 2008 — Chang, F., Dean, J., Ghemawat, S., Hsieh, W., Wallach, D., Burrows, M., Chandra, T., Fikes, A. and Gruber, R. (2008). Bigtable: A distributed storage system for structured data. *ACM Trans. on Computer Systems* Vol. 26, No. 2, pp. 1–26.

Charron-Bost 1991 — Charron-Bost, B. (1991). Concerning the size of logical clocks in distributed systems. *Information Processing Letter*s, Vol. 39, No.1, pp. 11–16.

Chaum 1981 — Chaum, D. (1981). Untraceable electronic mail, return addresses and digital pseudonyms. *Comms. ACM*, Vol. 24, No. 2, pp. 84–88.

Chen *et al.* 1994 — Chen, P., Lee, E., Gibson, G., Katz, R. and Patterson, D. (1994). RAID: High-performance, reliable secondary storage. *ACM Computing Surveys,* Vol. 26, No. 2, pp. 145–188.

Cheng 1998 — Cheng, C.K. (1998). *A survey of media servers*. Hong Kong University CSIS, November.

Cheng *et al*. 2005 — Cheng, Y-C., Chawathe, Y., LaMarca, A. and Krumm J. (2005) Accuracy characterization for metropolitan-scale Wi-Fi localization, *Third International Conference on Mobile Systems, Applications, and Services (MobiSys 2005)*, June.

Cheriton 1984 — Cheriton, D.R. (1984). The V kernel: A software base for distributed systems. *IEEE Software*, Vol. 1, No. 2, pp. 19–42.

Cheriton 1986 — Cheriton, D.R. (1986). VMTP: A protocol for the next generation of communication systems. In *Proceedings of the SIGCOMM '86 Symposium on Communication Architectures and Protocols*, pp. 406–15.

Cheriton and Mann 1989 — Cheriton, D. and Mann, T. (1989). Decentralizing a global naming service for improved performance and fault tolerance. *ACM Transactions on Computer Systems*, Vol. 7, No. 2, pp. 147–83.

Cheriton and Skeen 1993 — Cheriton, D. and Skeen, D. (1993). Understanding the limitations of causally and totally ordered communication. In *Proceedings of the 14th ACM Symposium on Operating System Principles*, Dec., pp. 44–57.

Cheriton and Zwaenepoel 1985 — Cheriton, D.R. and Zwaenepoel, W. (1985). Distributed process groups in the V kernel. *ACM Transactions on Computer Systems*, Vol. 3, No. 2, pp. 77–107.

Cheswick and Bellovin 1994	Cheswick, E.R. and Bellovin, S.M. (1994). *Firewalls and Internet Security*. Reading, MA: Addison-Wesley.
Chien 2004	Chien, A. (2004). Massively distributed computing: Virtual screening on a desktop Grid. In Foster, I. and Kesselman, C. (eds.), *The Grid 2*. San Francisco, CA: Morgan Kauffman.
Chisnall 2007	Chisnall, D. (2007). *The Definitive Guide to the Xen Hypervisor*. Upper Saddle River, NJ: Prentice-Hall.
Choudhary *et al.* 1989	Choudhary, A., Kohler, W., Stankovic, J. and Towsley, D. (1989). A modified priority based probe algorithm for distributed deadlock detection and resolution. *IEEE Transactions Software Engineering*, Vol. 15, No. 1, pp. 10–17.
Chu *et al.* 2000	Chu,Y.-H., Rao, S.G. and Zhang, H. (2000). A case for end system multicast. In *Proc. of ACM Sigmetrics*, June, pp. 1–12.
Cilia *et al*. 2004	Cilia, M., Antollini, M., Bornhövd, C. and Buchmann, A. (2004). Dealing with heterogeneous data in pub/sub systems: The concept-based approach. In *Proceedings of the International Workshop on Distributed Event-Based Systems*, Edinburgh, Scotland, May, pp. 26–31.
Clark 1982	Clark, D.D. (1982). *Window and Acknowledgement Strategy in TCP*. Internet RFC 813.
Clark 1988	Clark, D.D. (1988). The design philosophy of the DARPA Internet protocols. *ACM SIGCOMM Computer Communication Review*, Vol. 18, No. 4, pp. 106–114.
Clarke *et al.* 2000	Clarke, I., Sandberg, O., Wiley, B. and Hong, T. (2000). Freenet: A distributed anonymous information storage and retrieval system. In *Proc. of the ICSI Workshop on Design Issues in Anonymity and Unobservability*, Berkeley, CA, pp. 46–66.
code.google.com I	Protocol buffers. *Home page*.
code.google.com II	Protocol buffers. *Developer guide: techniques*.
code.google.com III	Protocol buffers. *Third-party add-ons (RPC implementations)*.
code.google.com IV	Google App Engine. *Home page.*
Cohen 2003	Cohen, B. (2003). Incentives build robustness in BitTorrent. May 2003, Internet publication.
Comer 2006	Comer, D.E. (2006). *Internetworking with TCP/IP, Volume 1: Principles, Protocols and Architecture*, 5th edn. Upper Saddle River, NJ: Prentice-Hall.
Comer 2007	Comer, D.E. (2007). *The Internet Book*, 4th edn. Upper Saddle River, NJ: Prentice-Hall.

Condict *et al*. 1994 — Condict, M., Bolinger, D., McManus, E., Mitchell, D. and Lewontin, S. (1994). *Microkernel modularity with integrated kernel performance*. Technical report, OSF Research Institute, Cambridge, MA, April.

Coulouris *et al.* 1998 — Coulouris, G.F., Dollimore, J. and Roberts, M. (1998). Role and task-based access control in the PerDiS groupware platform. *Third ACM Workshop on Role-Based Access Control*, George Mason University, Washington, DC, October 22–23.

Coulson *et al*. 2008 — Coulson, G., Blair, G., Grace, P., Taiani, F., Joolia, A., Lee, K., Ueyama, J. and Sivaharan, T. (2008). A generic component model for building systems software. *ACM Trans. on Computer Syst*ems, Vol. 26, No. 1, pp. 1–42.

Cristian 1989 — Cristian, F. (1989). Probabilistic clock synchronization. *Distributed Computing*, Vol. 3, pp. 146–58.

Cristian 1991 — Cristian, F. (1991). Reaching agreement on processor group membership in synchronous distributed systems. *Distributed Computing*, Vol. 4, pp. 175–87.

Cristian and Fetzer 1994 — Cristian, F. and Fetzer, C. (1994). Probabilistic internal clock synchronization. In *Proceedings of the 13th Symposium on Reliable Distributed Systems*, October, pp. 22–31.

Crow *et al.* 1997 — Crow, B., Widjaja, I., Kim, J. and Sakai, P. (1997). IEEE 802.11 wireless local area networks. *IEEE Communications Magazine*, Vol. 35, No. 9, pp. 116–26.

cryptography.org — *North American Cryptography Archives*.

Culler *et al*. 2001 — Culler, D.E., Hill, J., Buonadonna, P., Szewczyk, R. and Woo, A. (2001). A network-centric approach to embedded software for tiny devices. *Proceedings of the First International Workshop on Embedded Software*, Tahoe City, CA, October, pp. 114–130.

Culler *et al*. 2004 — Culler, D., Estrin, D. and Srivastava, M. (2004). Overview of sensor networks. *IEEE Computer*, Vol. 37, No. 8, pp. 41–49.

Curtin and Dolske 1998 — Kurtin, M. and Dolski, J. (1998). A brute force search of DES Keyspace. *;login: – the Newsletter of the USENIX Association*, May.

Custer 1998 — Custer, H. (1998). *Inside Windows NT*, 2nd edn.. Redmond, WA: Microsoft Press.

Czerwinski *et al.* 1999 — Czerwinski, S., Zhao, B., Hodes, T., Joseph, A. and Katz, R. (1999). An architecture for a secure discovery service. In *Proceedings of the Fifth Annual International Conference on Mobile Computing and Networks*. Seattle, WA, pp.24–53.

Dabek *et al.* 2001 — Dabek, F., Kaashoek, M.F., Karger, D., Morris, R. and Stoica, I. (2001). Wide-area cooperative storage with CFS. In *Proc. of the ACM Symposium on Operating System Principles*, October, pp. 202–15.

Dabek *et al*. 2003 — Dabek, F., Zhao, B., Druschel, P., Kubiatowicz, J. and Stoica, I.(2003). Ion Stoica, Towards a common API for structured peer-to-peer overlays. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, Berkeley, CA, February, pp. 33–44.

Daemen and Rijmen 2000 — Daemen, J. and Rijmen, V. (2000). The block cipher Rijndael. Quisquater, J.-J. and Schneier, B.(eds.). Smart Card Research and Applications, LNCS 1820. Springer-Verlag, pp. 288–296.

Daemen and Rijmen 2002 — Daemen, J. and Rijmen, V. (2002). *The Design of Rijndael: AES – The Advanced Encryption Standard*, New York: Springer-Verlag.

Dasgupta *et al.* 1991 — Dasgupta, P., LeBlanc Jr., R.J., Ahamad, M. and Ramachandran, U. (1991). The Clouds distributed operating system. *IEEE Computer*, Vol. 24, No. 11, pp. 34–44.

Davidson 1984 — Davidson, S.B. (1984). Optimism and consistency in partitioned database systems. *ACM Transactions on Database Systems*, Vol. 9, No. 3, pp. 456–81.

Davidson *et al.* 1985 — Davidson, S.B., Garcia-Molina, H. and Skeen, D. (1985). Consistency in partitioned networks. *Computing Surveys*, Vol. 17, No. 3, pp. 341–70.

Davies *et al*. 1998 — Davies, N., Friday, A., Wade, S. and Blair, G. (1998). L^2imbo: a distributed systems platform for mobile computing. *Mobile Networks and Applications*, Vol. 3, No. 2, pp. 143–156.

de Ipiña *et al*. 2002 — de Ipiña, D.L., Mendonça, P. and Hopper, A. (2002). TRIP: A low-cost vision-based location system for ubiquitous computing. *Personal and Ubiquitous Computing*, Vol. 6, No. 3, pp. 206–219.

Dean 2006 — Dean, J. (2006) Experiences with MapReduce, an abstraction for large-scale computation. In *Proc. of the 15th International Conference on Parallel Architectures and Compilation Techniques*, (PACT'06), Seattle, WA, p. 1.

Dean and Ghemawat 2004 — Dean, J. and Ghemawat, S. (2004). MapReduce: simplified data processing on large clusters. In *Proc. of Operating System Design and Implementation*, (OSDI'04), San Francisco, CA, pp. 137–150.

Dean and Ghemawat 2008 — Dean, J. and Ghemawat, S. (2008). MapReduce: Simplified data processing on large clusters. *Comms. ACM*, Vol. 51, No. 1, pp. 107–113.

Dean and Ghemawat 2010 — Dean, J. and Ghemawat, S. (2010). MapReduce: a flexible data processing tool. *Comms. ACM*, Vol. 53, No. 1, pp. 72–77.

Debaty and Caswell 2001 — Debaty, P. and Caswell, D. (2001). Uniform web presence architecture for people, places, and things. *IEEE Personal Communications,* Vol. 8, No. 4, pp. 6–11.

DEC 1990 — Digital Equipment Corporation (1990). *In Memoriam: J. C. R. Licklider 1915–1990*. Technical Report 61, DEC Systems Research Center.

DeCandia *et al.* 2007 — DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P. and Vogels, W. (2007). Dynamo: Amazon's highly available key-value store. *SIGOPS Oper. Syst. Rev*. Vol. 41, No. 6, pp. 205–220.

Delgrossi *et al.* 1993 — Delgrossi, L., Halstrick, C., Hehmann, D., Herrtwich, R.G., Krone, O., Sandvoss, J. and Vogt, C. (1993). Media scaling for audiovisual communication with the Heidelberg transport system. *ACM Multimedia '93*, Anaheim, CA.

Demers *et al.* 1989 — Demers, A., Keshav, S. and Shenker, S. (1989). Analysis and simulation of a fair queueing algorithm. *ACM SIGCOMM '89*.

Denning and Denning 1977 — Denning, D. and Denning, P. (1977). Certification of programs for secure information flow. *Comms. ACM*, Vol. 20, No. 7, pp. 504–13.

Dertouzos 1974 — Dertouzos, M.L. (1974). Control robotics – the procedural control of physical processes. *IFIP Congress*.

Dierks and Allen 1999 — Dierks, T. and Allen, C. (1999). *The TLS Protocol Version 1.0*. Internet RFC 2246.

Diffie 1988 — Diffie, W. (1988). The first ten years of public-key cryptography. *Proceedings of the IEEE*, Vol. 76, No. 5, pp. 560–77.

Diffie and Hellman 1976 — Diffie, W. and Hellman, M.E. (1976). New directions in cryptography. *IEEE Transactions Information Theory*, Vol. IT-22, pp. 644–54.

Diffie and Landau 1998 — Diffie, W. and Landau, S. (1998). *Privacy on the Line*. Cambridge, MA: MIT Press.

Dijkstra 1959 — Dijkstra, E.W. (1959). A note on two problems in connection with graphs. *Numerische Mathematic*, Vol. 1, pp. 269–71.

Dilley *et al.* 2002 — Dilley, J., Maggs, B., Parikh, J., Prokop, H., Sitaraman, R. and Weihl, B. (2002). Globally distributed content delivery. *IEEE Internet Computin*g, pp. 50–58.

Dingledine *et al.* 2000 — Dingledine, R., Freedman, M.J. and Molnar, D. (2000). The Free Haven project: Distributed anonymous storage service. In *Proc. Workshop on Design Issues in Anonymity and Unobservability*, Berkeley, CA, July, pp. 67–95.

Dolev and Malki 1996 — Dolev, D. and Malki, D. (1996). The Transis approach to high availability cluster communication. *Comms. ACM*, Vol. 39, No. 4, pp. 64–70.

Dolev and Strong 1983 — Dolev, D. and Strong, H. (1983). Authenticated algorithms for Byzantine agreement. *SIAM Journal of Computing*, Vol. 12, No. 4, pp. 656–66.

Dolev *et al.* 1986 — Dolev, D., Halpern, J., and Strong, H. (1986). On the possibility and impossibility of achieving clock synchronization. *Journal of Computing Systems Science*, Vol. 32, No. 2 , pp. 230–50.

Dorcey 1995 — Dorcey, T. (1995). CU-SeeMe desktop video conferencing software. *Connexions*, Vol. 9, No. 3, pp. 42–45.

Douceur and Bolosky 1999 — Douceur, J.R. and Bolosky, W. (1999). Improving responsiveness of a stripe-scheduled media server. In Proc. IS &T/SPIE Conf. on *Multimedia Computing and Networking*, pp. 192–203.

Douglis and Ousterhout 1991 — Douglis, F. and Ousterhout, J. (1991). Transparent process migration: Design alternatives and the Sprite implementation, *Software – Practice and Experience*, Vol. 21, No. 8, pp. 757–89.

Draves 1990 — Draves, R. (1990). A revised IPC interface. In *Proceedings of the USENIX Mach Workshop*, pp. 101–21, October.

Draves *et al.* 1989 — Draves, R.P., Jones, M.B. and Thompson, M.R. (1989). *MIG - the Mach Interface Generator*. Technical Report, Dept. of Computer Science, Carnegie-Mellon University.

Druschel and Peterson 1993 — Druschel, P. and Peterson, L. (1993). Fbufs: a high-bandwidth cross-domain transfer facility. In *Proceedings of the 14th ACM Symposium on Operating System Principles*, pp. 189–202.

Druschel and Rowstron 2001 — Druschel, P. and Rowstron, A. (2001). PAST: A large-scale, persistent peeer-to-peer storage utility. In *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, Schloss Elmau, Germany, May, pp. 75–80.

Dubois *et al.* 1988 — Dubois, M., Scheurich, C. and Briggs, F.A. (1988). Synchronization, coherence and event ordering in multiprocessors. *IEEE Computer*, Vol. 21, No. 2, pp. 9–21.

Dwork *et al.* 1988 — Dwork, C., Lynch, N. and Stockmeyer, L. (1988). Consensus in the presence of partial synchrony. *Journal of the ACM*, Vol. 35, No. 2, pp. 288–323.

Eager *et al.* 1986 — Eager, D., Lazowska, E. and Zahorjan, J. (1986). Adaptive load sharing in homogeneous distributed systems. *IEEE Transactions on Software Engineering*, Vol. SE-12, No. 5, pp. 662–675.

earth.google.com — Google Earth. *Home page*.

Edney and Arbaugh 2003 — Edney, J. and Arbaugh, W. (2003). *Real 802.11 Security: Wi-Fi Protected.* Boston MA: Pearson Education.

Edwards and Grinter 2001 — Edwards, W.K. and Grinter, R. (2001). At home with ubiquitous computing: Seven challenges. In *Proceedings of the Third International Conference on Ubiquitous Computing (Ubicomp 2001)*, Atlanta, GA, Sep.–Oct., pp. 256–272.

Edwards *et al*. 2002 — Edwards, W.K., Newman, M.W., Sedivy, J.Z., Smith, T.F. and Izadi, S. (2002). Challenge: Recombinant computing and the speakeasy approach. In *Proceedings of the Eighth ACM International Conference on Mobile Computing and Networking (MobiCom 2002)*, Atlanta, GA, September, pp. 279–286.

EFF 1998 — Electronic Frontier Foundation (1998). *Cracking DES, Secrets of Encryption Research, Wiretap Politics & Chip Design*. Sebastapol, CA: O'Reilly & Associates.

Egevang and Francis 1994 — Egevang, K. and Francis, P. (1994). *The IP Network Address Translator (NAT)*. Internet RFC 1631.

Eisler *et al.* 1997 — Eisler, M., Chiu, A. and Ling, L. (1997). *RPCSEC_GSS Protocol Specification*. Sun Microsystems, Internet RFC 2203.

El Abbadi *et al.* 1985 — El Abbadi, A., Skeen, D. and Cristian, C. (1985). An efficient fault-tolerant protocol for replicated data management. In *Proc. of the 4th Annual ACM SIGACT/SIGMOD Symposium on Principles of Data Base Systems*, Portland, OR, pp. 215–29.

Ellis *et al.* 1991 — Ellis, C., Gibbs, S. and Rein, G. (1991). Groupware – some issues and experiences. *Comms. ACM*, Vol. 34, No. 1, pp. 38–58.

Ellison 1996 — Ellison, C. (1996). Establishing identity without certification authorities. In *Proc. of the 6th USENIX Security Symposium*, San Jose, CA, July, p.7.

Ellison *et al.* 1999 — Ellison, C., Frantz, B., Lampson, B., Rivest, R., Thomas, B. and Ylonen, T. (1999). *SPKI Certificate Theory*. Internet RFC 2693, September.

Elrad *et al*. 2001 — Elrad, T., Filman, R. and Bader A. (eds.) (2001). Theme section on aspect-oriented programming, *Comms. ACM*, Vol. 44, No. 10, pp. 29–32.

Emmerich 2000 — Emmerich, W. (2000). *Engineering Distributed Objects*. New York: John Wiley & Sons.

esm.cs.cmu.edu — ESM project at CMU. *Home page*

Eugster *et al*. 2003 — Eugster, P.T., Felber, P.A., Guerraoui, R. and Kermarrec A-M. (2003). The many faces of publish-subscribe, *ACM Computing Surveys*, Vol. 35, No. 2, pp. 114–131.

Evans *et al.* 2003 — Evans, C. and 15 other authors (2003). *Web Services Reliability (WS-Reliability)*, Fujitsu, Hitachi, NEC, Oracle Corporation, Sonic Software, and Sun Microsystems.

Fall 2003 — Fall, K. (2003). A delay-tolerant network architecture for challenged internets. In *Proceedings of the ACM 2003 Conference on Applications, Technologies, Architectures and Protocols for Computer Communications (SIGCOMM 2003)*, Karlsruhe, Germany, August, pp. 27–34.

Farley 1998 — Farley, J. (1998). *Java Distributed Computing*. Cambridge, MA: O'Reilly.

Farrow 2000 — Farrow, R. (2000). Distributed denial of service attacks – how Amazon, Yahoo, eBay and others were brought down. *Network Magazine*, March.

Ferguson and Schneier 2003 — Ferguson, N. and Schneier, B. (2003). *Practical Cryptography*. New York: John Wiley & Sons.

Ferrari and Verma 1990 — Ferrari, D. and Verma, D. (1990). A scheme for real-time channel establishment in wide-area networks. *IEEE Journal on Selected Areas in Communications*, Vol. 8, No. 4, pp. 368–79.

Ferreira *et al.* 2000 — Ferreira, P., Shapiro, M., Blondel, X., Fambon, O., Garcia, J., Kloostermann, S., Richer, N., Roberts, M., Sandakly, F., Coulouris, G., Dollimore, J., Guedes, P., Hagimont, D. and Krakowiak, S. (2000). PerDiS: Design, implementation, and use of a PERsistent DIstributed Store. In *LNCS 1752: Advances in Distributed Systems*. Berlin, Heidelberg, New York: Springer-Verlag. pp. 427–53.

Ferris and Langworthy 2004 — Ferris, C. and Langworthy, D. (eds.), Bilorusets, R. and 22 other authors (2004). *Web Services Reliable Messaging Protocol (WS-Reliable Messaging)*. BEA, IBM, Microsoft and TibCo.

Fidge 1991 — Fidge, C. (1991). Logical time in distributed computing systems. *IEEE Computer*, Vol. 24, No. 8, pp. 28–33.

Fielding 2000 — Fielding, R. (2000). *Architectural Styles and the Design of Network-based Software Architecture*s. Doctoral dissertation, University of California, Irvine

Fielding *et al.* 1999 — Fielding, R., Gettys, J., Mogul, J.C., Frystyk, H., Masinter, L., Leach, P. and Berners-Lee, T. (1999). *Hypertext Transfer Protocol – HTTP/1.1*. Internet RFC 2616.

Filman *et al*. 2004 — Filman, R., Elrad, T., Clarke, S. and Aksit, M. (2004) *Aspect-Oriented Software Development*. Addison-Wesley.

Fischer 1983 — Fischer, M. (1983). The consensus problem in unreliable distributed systems (a brief survey). In Karpinsky, M. (ed.), *Foundations of Computation Theory*, Vol. 158 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 127–140. Yale University Technical Report YALEU/DCS/TR-273.

Fischer and Lynch 1982 — Fischer, M. and Lynch, N. (1982). A lower bound for the time to assure interactive consistency. *Inf. Process. Letters*, Vol. 14, No. 4, pp. 183–6.

Fischer and Michael 1982 — Fischer, M.J. and Michael, A. (1982). Sacrificing serializability to attain high availability of data in an unreliable network. In *Proceedings of the Symposium on Principles of Database Systems*, pp. 70–5.

Fischer *et al.* 1985 — Fischer, M., Lynch, N. and Paterson, M. (1985). Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, Vol. 32, No. 2, pp. 374–82.

Fitzgerald and Rashid 1986 — Fitzgerald, R. and Rashid, R.F. (1986). The integration of virtual memory management and interprocess communication in Accent. *ACM Transactions on Computer Systems*, Vol. 4, No. 2, pp. 147–77.

Flanagan 2002	Flanagan, D. (2002). *Java in a Nutshell*, 4th edn. Cambridge, MA: O'Reilly.
Floyd 1986	Floyd, R. (1986). *Short term file reference patterns in a UNIX environment*. Technical Rep. TR 177, Rochester, NY: Dept of Computer Science, University of Rochester.
Floyd and Jacobson 1993	Floyd, S. and Jacobson, V. (1993). The synchronization of periodic routing messages. *ACM Sigcomm '93 Symposium*.
Floyd *et al.* 1997	Floyd, S., Jacobson, V., Liu, C., McCanne, S. and Zhang, L. (1997). A reliable multicast framework for lightweight sessions and application level framing. *IEEE/ACM Transactions on Networking*, Vol. 5, No. 6, pp. 784–803.
Fluhrer *et al.* 2001	Fluhrer, S., Mantin, I. and Shamir, A. (2001). Weaknesses in the key scheduling algorithm of RC4. In *Proceedings of the 8th annual workshop on Selected Areas of Cryptography (SAC)*, Toronto, Canada, pp. 1–24.
Ford and Fulkerson 1962	Ford, L.R. and Fulkerson, D.R. (1962). *Flows in Networks*. Princeton, NJ: Princeton University Press.
Foster and Kesselman 2004	Foster, I. and Kesselman, C. (eds.) (2004). *The Grid 2*. San Francisco, CA: Morgan Kauffman.
Foster *et al.* 2001	Foster, I., Kesselman, C. and Tuecke, S. (2001). The anatomy of the Grid: Enabling scalable virtual organisations. *Intl. J. Supercomputer Applications*, Vol. 15, No. 3, pp. 200–222.
Foster *et al.* 2002	Foster, I., Kesselman, C., Nick, J. and Tuecke, S. (2002). Grid services for distributed systems integration. *IEEE Computer*, Vol. 35, No. 6, pp. 37–46.
Foster *et al.* 2004	Foster, I., Kesselman, C. and Tuecke, S. (2004). T*he Open Grid Services Architecture*. In Foster, I. and Kesselman, C. (eds.), *The Grid 2*. San Francisco, CA: Morgan Kauffman.
Fox *et al.* 1997	Fox, A., Gribble, S., Chawathe, Y., Brewer, E. and Gauthier, P. (1997). Cluster-based scalable network services. *Proceedings of the 16th ACM Symposium on Operating Systems Principles*, pp. 78–91.
Fox *et al*. 1998	Fox, A., Gribble, S.D., Chawathe, Y. and Brewer, E.A. (1998). Adapting to network and client variation using active proxies: Lessons and perspectives. *IEEE Personal Communications*, Vol. 5, No. 4, pp. 10–19.
Fox *et al*. 2003	Fox, D., Hightower, J., Liao, L., Schulz, D. and Borriello, G. (2003). Bayesian filtering for location estimation. *IEEE Pervasive Computing*, Vol. 2, No. 3, pp. 24–33.
fractal.ow2.org I	Fractal Project. *Home page*.
fractal.ow2.org II	Fractal Project. *Tutorial*.

France and Rumpe 2007 — France, R. and Rumpe, B. (2007). Model-driven development of complex software: A research roadmap. *International Conference on Software Engineering (Future of Software Engineering session)*. IEEE Computer Society, Washington, DC, pp. 37–54.

Fraser *et al*. 2003 — Fraser, K.A., Hand, S.M., Harris, T.L., Leslie, I.M. and Pratt, I.A. (2003). *The Xenoserver computing infrastructure.* Technical Report UCAM-CL-TR-552, Computer Laboratory, University of Cambridge.

Freed and Borenstein 1996 — Freed, N. and Borenstein, N. (1996). *MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies. September*. Internet RFC 1521.

freenet.project.org — The Free Network Project.

freepastry.org — The FreePastry project.

Frey and Roman 2007 — Frey, D. and G-C Roman, G-C. (2007). Context-aware publish subscribe in mobile ad hoc networks. In *Proceedings of the 9th International Conference on Coordination Models and Languages*, pp. 37–55.

Ganesh *et al*. 2003 — Ganesh, A. J., Kermarrec, A. and Massoulié, L. (2003). Peer-to-peer membership management for gossip-based protocols. *IEEE Transactions on Computers*, Vol. 52, No. 2, pp. 139–149.

Garay and Moses 1993 — Garay, J. and Moses, Y. (1993). Fully polynomial Byzantine agreement in t+1 rounds. In *Proceedings of the 25th ACM symposium on theory of computing*, pp. 31–41.

Garcia-Molina 1982 — Garcia-Molina, H. (1982). Elections in distributed computer systems. *IEEE Transactions on Computers*, Vol. C-31, No. 1, pp. 48–59.

Garcia-Molina and Spauster 1991 — Garcia-Molina, H. and Spauster, A. (1991). Ordered and reliable multicast communication. *ACM Transactions on Computer Systems*, Vol. 9, No. 3, pp. 242–71.

Garfinkel 1994 — Garfinkel, S. (1994). *PGP: Pretty Good Privacy*. Cambridge, MA: O'Reilly.

Gehrke and Madden 2004 — Gehrke, J. and Madden, S. (2004). Query processing in sensor networks. *IEEE Pervasive Computing*, Vol. 3, No. 1, pp. 46–55.

Gelernter 1985 — Gelernter, D. (1985). Generative communication in Linda. *ACM Transactions on Programming Languages and Systems*, Vol. 7, No. 1, pp. 80–112.

Ghemawat *et al*. 2003 — Ghemawat, S., Gobioff, H. and Leung, S. (2003). The Google file system. *SIGOPS Oper. Syst. Rev*., Vol. 37, No. 5, pp. 29–43.

Gibbs and Tsichritzis 1994 — Gibbs, S.J. and Tsichritzis, D.C. (1994). *Multimedia Programming*. Addison-Wesley.

Gibson *et al*. 2004	Gibson, G., Courtial, J., Padgett, M.J., Vasnetsov, M., Pas'ko, V., Barnett, S.M. and Franke-Arnold, S. (2004). Free-space information transfer using light beams carrying orbital angular momentum. *Optics Express*, Vol. 12, No. 22, pp. 5448–5456.
Gifford 1979	Gifford, D.K. (1979). Weighted voting for replicated data. In *Proceedings of the 7th Symposium on Operating Systems Principles*, pp. 150–62.
glassfish.dev.java.net	GlassFish Application Server. *Home page.*
Gokhale and Schmidt 1996	Gokhale, A. and Schmidt, D. (1996). Measuring the performance of communication middleware on high-speed networks. In *Proceedings of SIGCOMM '96*, pp. 306–17.
Golding and Long 1993	Golding, R. and Long, D. (1993). *Modeling replica divergence in a weak-consistency protocol for global-scale distributed databases*. Technical report UCSC-CRL-93-09, Computer and Information Sciences Board, University of California, Santa Cruz.
Goldschlag *et al.* 1999	Goldschlag, D., Reed, M. and Syverson, P. (1999). Onion routing for anonymous and private Internet connections. *Communications of the ACM*, Vol. 42, No. 2, pp. 39–41.
Golub *et al.* 1990	Golub, D., Dean, R., Forin, A. and Rashid, R. (1990). *UNIX as an application program.* In *Proc. USENIX Summer Conference*, pp. 87–96.
Gong 1989	Gong, L. (1989). A secure identity-based capability system. In *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, May, pp. 56–63.
googleblog.blogspot.com I	The Official Google Blog. *Powering a Google search.*
googleblog.blogspot.com II	The Official Google Blog. *New Search Engine: Caffeine*.
googletesting.blogspot.com	The Google Testing Blog. *Home page*.
Gordon 1984	Gordon, J. (1984). *The Story of Alice and Bob*. After dinner speech, see also: en.wikipedia.org/wiki/Alice_and_Bob .
Govindan and Anderson 1991	Govindan, R. and Anderson, D.P. (1991). Scheduling and IPC mechanisms for continuous media. *ACM Operating Systems Review,* Vol. 25, No. 5, pp. 68–80.
Goyal and Carter 2004	Goyal, S. and Carter, J. (2004). A lightweight secure cyber foraging infrastructure for resource-constrained devices. In *Proceedings of the Sixth IEEE Workshop on Mobile Computing Systems and Applications* (WMCSA 2004), December, pp. 186–195.

Graumann *et al*. 2003 — Graumann, D., Lara, W., Hightower, J. and Borriello, G. (2003). Real-world implementation of the Location Stack: The Universal Location Framework. In *Proceedings of the 5th IEEE Workshop on Mobile Computing Systems & Applications (WMCSA 2003)*, Monterey, CA, October, pp. 122–128.

Grace *et al*. 2003 — Grace, P., Blair, G.S. and Samuel, S. (2003). ReMMoC: A reflective middleware to support mobile client interoperability. In *Proceedings of the International Symposium on Distributed Objects and Applications (DOA'03)*, Catania, Sicily, November, pp. 1170–1187.

Grace *et al*. 2008 — Grace, P., Hughes, D., Porter, B., Blair, G.S., Coulson, G. and Taiani, F. (2008). Experiences with open overlays: A middleware approach to network heterogeneity. In *Proceedings of the 3rd ACM Sigops/Eurosys European Conference on Computer Systems (Eurosys'08)*, Glasgow, Scotland, pp. 123–136.

Gray 1978 — Gray, J. (1978). Notes on database operating systems. In *Operating Systems: an Advanced Course*. *Lecture Notes in Computer Science*, Vol. 60, Springer-Verlag, pp. 394–481.

Gray and Szalay 2002 — Gray, J. and Szalay, A. (2002). *The World-Wide Telescope, an Archetype for Online Science*. Technical Report. MSR-TR-2002-75. Microsoft Research.

Greenfield and Dornan 2004 — Greenfield, D. and Dornan, A. (2004). Amazon: Web site to web services, *Network Magazine*, October.

Grimm 2004 — Grimm, R. (2004). One.world: Experiences with a pervasive computing architecture. *IEEE Pervasive Computing*, Vol. 3, No. 3, pp. 22–30.

Gruteser and Grunwald 2003 — Gruteser, M. and Grunwald, D. (2003). Enhancing location privacy in wireless LAN through disposable interface identifiers: a quantitative analysis. In *Proceedings of the 1st ACM international workshop on Wireless mobile applications and services on WLAN hotspots (WMASH '03)*, San Diego, CA, September, pp. 46–55.

Guerraoui *et al.* 1998 — Guerraoui, R., Felber, P., Garbinato, B. and Mazouni, K. (1998). System support for object groups. In *Proceedings of the ACM Conference on Object Oriented Programming Systems, Languages and Applications* (OOPSLA'98).

Gummadi *et al.* 2003 — Gummadi, K.P., Gummadi, R., Gribble, S.D., Ratnasamy, S., Shenker, S. and Stoica, I. (2003). The impact of DHT routing geometry on resilience and proximity. In Proc. *ACM SIGCOMM 2003,* pp. 381–94.

Gupta *et al.* 2004 — Gupta, A., Sahin, O. D., Agrawal, D. and Abbadi, A. E. (2004). Meghdoot: Content-based publish/subscribe over P2P networks. In *Proceedings of the 5th ACM/IFIP/USENIX International Conference on Middleware,* Toronto, Canada, October, pp. 254–273.

Gusella and Zatti 1989 — Gusella, R. and Zatti, S. (1989). The accuracy of clock synchronization achieved by TEMPO in Berkeley UNIX 4.3BSD. *IEEE Transactions on Software Engineering*, Vol. 15, No. 7, pp. 847–53.

Guttman 1999 — Guttman, E. (1999). Service location protocol: Automatic discovery of IP network services. *IEEE Internet Computing*, Vol. 3, No. 4, pp. 71–80.

Haartsen *et al*. 1998 — Haartsen, J., Naghshineh, M., Inouye, J., Joeressen, O.J. and Allen, W. (1998). Bluetooth: Vision, goals and architecture. *ACM Mobile Computing and Communications Review*, Vol. 2, No. 4, pp. 38–45.

hadoop.apache.org — Hadoop. *Home page*.

Hadzilacos and Toueg 1994 — Hadzilacos, V. and Toueg, S. (1994). *A Modular Approach to Fault-tolerant Broadcasts and Related Problems*. Technical report, Dept of Computer Science, University of Toronto.

Hand *et al.* 2003 — Hand, S., Harris, T., Kotsovinos, E. and Pratt, I. (2003). Controlling the XenoServer open platform. In *Proceedings of the 6th IEEE Conference on Open Architectures and Network Programming (OPEN ARCH 2003)*, pp. 3–11.

Handley *et al.* 1999 — Handley, M., Schulzrinne, H., Schooler, E. and Rosenberg, J. (1999). *SIP: Session Initiation Protocol*. Internet RFC 2543.

Harbison 1992 — Harbison, S. P. (1992). *Modula-3*. Englewood Cliffs, NJ: Prentice-Hall.

Härder 1984 — Härder, T. (1984). Observations on optimistic concurrency control schemes. *Information Systems*, Vol. 9, No. 2, pp. 111–20.

Härder and Reuter 1983 — Härder, T. and Reuter, A. (1983). Principles of transaction-oriented database recovery. *ACM Computing Surveys*, Vol. 15, No. 4, pp. 287–317.

Harrenstien et al. 1985 — Harrenstien, K., Stahl, M. and Feinler, E. (1985). *DOD Internet Host Table Specification*. Internet RFC 952.

Harter and Hopper 1994 — Harter, A. and Hopper, A. (1994). A distributed location system for the active office. *IEEE Network*, Vol. 8, No. 1, pp. 62–70.

Harter *et al*. 2002 — Harter, A., Hopper, A., Steggles, P., Ward, A. and Webster, P. (2002). The anatomy of a context-aware application. *Wireless Networks*, Vol. 8, No. 2–3, pp. 187–197.

Härtig *et al.* 1997 — Härtig, H., Hohmuth, M., Liedtke, J., Schönberg, S. and Wolter, J. (1997). The performance of kernel-based systems. In *Proceedings of the 16th ACM Symposium on Operating System Principles*, pp. 66–77.

Hartman and Ousterhout 1995 — Hartman, J. and Ousterhout, J. (1995). The Zebra striped network file system. *ACM Trans. on Computer Systems*, Vol. 13 , No. 3, pp. 274–310.

Hauch and Reiser 2000 — Hauch, R. and Reiser, H. (2000). Monitoring quality of service across organisational boundaries. In *Trends in Distributed Systems: Towards a Universal Service Market*, Proc. Third Intl. IFIP/HGI Working conference, USM, September.

Hayton *et al.* 1998 — Hayton, R., Bacon, J. and Moody, K. (1998). OASIS: Access control in an open, distributed environment. In *Proceedings of the IEEE Symposium on Security and Privacy*, May, Oakland, CA, pp. 3–14.

Hedrick 1988 — Hedrick, R. (1988). *Routing Information Protocol*. Internet RFC 1058.

Heidemann *et al*. 2001 — Heidemann, J., Silva, F., Intanagonwiwat, C., Govindan, R., Estrin, D. and Ganesan, D. (2001). Building efficient wireless sensor networks with low-level naming. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles*, Banff, Alberta, Canada, October, pp. 146–159.

Heineman and Councill 2001 — Heineman, G.T. and Councill, W.T. (2001). *Component-Based Software Engineering: Putting the Pieces Together*. Reading, MA: Addison-Wesley.

Hennessy and Patterson 2006 — Hennessy, J.L. and Patterson, D.A. (2006). *Computer Architecture: A Quantitative Approach*, 4th edn. San Francisco:CA: Morgan Kaufmann.

Henning 1998 — Henning, M. (1998). Binding, migration and scalability in CORBA. *Comms. ACM*, Vol. 41, No. 10, pp. 62–71.

Henning and Vinoski 1999 — Henning, M. and Vinoski, S. (1999). *Advanced CORBA Programming with C++*. Reading, MA: Addison-Wesley.

Herlihy 1986 — Herlihy, M. (1986). A quorum-consensus replication method for abstract data yypes. *ACM Transactions on Computer Systems*, Vol. 4, No. 1, pp. 32–53.

Herlihy and Wing 1990 — Herlihy, M. and Wing, J. (1990). On linearizability: a correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems*, Vol. 12, No. 3, pp. 463–92.

Herrtwich 1995 — Herrtwich, R.G. (1995). Achieving quality of service for multimedia applications. *ERSADS '95, European Research Seminar on Advanced Distributed Systems*, l'Alpe d'Huez, France, April.

Hightower and Borriello 2001 — Hightower, J. and Borriello, G. (2001). Location systems for ubiquitous computing. *IEEE Computer*, Vol. 34, No. 8, pp. 57–66.

Hightower *et al*. 2002 — Hightower, J., Brumitt, B. and Borriello, G. (2002). The Location Stack: A layered model for location in ubiquitous computing. In *Proceedings of the 4th IEEE Workshop on Mobile Computing Systems & Applications (WMCSA 2002)*, Callicoon, NY, June, pp. 22–28.

Hill *et al*. 2000 — Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D. and Pister, K. (2000). System architecture directions for networked sensors. In *Proceedings of the Ninth ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-IX)*, Cambridge, MA, November, pp. 93–104.

Hirsch 1997 — Hirsch, F.J. (1997). Introducing SSL and Certificates using SSLeay. *World Wide Web Journal*, Vol. 2, No. 3, pp. 141–173.

Holmquist *et al*. 2001 — Holmquist, L.E., Mattern, F., Schiele, B., Alahuhta, P., Beigl, M. and Gellersen, H.-W. (2001). Smart-Its Friends: A technique for users to easily establish connections between smart artefacts. In *Proceedings of the Third International Conference on Ubiquitous Computing (Ubicomp 2001)*, Atlanta, GA, September –October, pp. 116–122.

Housley 2002 — Housley, R. (2002). *Cryptographic Message Syntax (CMS) Algorithms*. Internet RFC 3370.

Howard *et al.* 1988 — Howard, J.H., Kazar, M.L., Menees, S.G, Nichols, D.A., Satyanarayanan, M., Sidebotham, R.N. and West, M.J. (1988). Scale and performance in a distributed file system. *ACM Transactions on Computer Systems*, Vol. 6, No. 1, pp. 51–81.

Huang *et al.* 2000 — Huang, A., Ling, B., Barton, J. and Fox, A. (2000). Running the Web backwards: Appliance data services. In *Proceedings of the 9th international World Wide Web Conference,* pp.619–31.

Huitema 1998 — Huitema, C. (1998). *IPv6 – the New Internet Protocol*. Upper Saddle River, NJ: Prentice-Hall.

Huitema 2000 — Huitema, C. (2000). *Routing in the Internet*, 2nd edn. Englewood Cliffs, NJ: Prentice-Hall.

Hull *et al*. 2004 — Hull, R., Clayton, B. and Melamad, T. (2004). Rapid authoring of mediascapes. In *Proceedings of the Sixth International Conference on Ubiquitous Computing (Ubicomp 2004)*, Nottingham, England, September, pp. 125–142.

hulu.com — Hulu. *Home page*.

Hunt *et al.* 2007 — Hunt, G.C. and Larus, J. R., Singularity: Rethinking the Software Stack, In *ACM SIGOPS Operating Systems Review*, Vol. 41, No. 2, pp. 37–49.

Hunter and Crawford 1998 — Hunter, J. and Crawford, W. (1998). *Java Servlet Programming*. Sebastopol, CA: O'Reilly.

Hutchinson and Peterson 1991 — Hutchinson, N. and Peterson, L. (1991). The x-kernel: An architecture for implementing network protocols. *IEEE Transactions on Software Engineering*, Vol. 17, No. 1, pp. 64–76.

Hutchinson *et al.* 1989 — Hutchinson, N.C., Peterson, L.L., Abbott, M.B. and O'Malley, S.W. (1989). RPC in the x-Kernel: Evaluating new design techniques. In *Proc. of the 12th ACM Symposium on Operating System Principles*, pp. 91–101.

Hyman *et al.* 1991 — Hyman, J., Lazar, A.A. and Pacifici, G. (1991). MARS – The MAGNET-II Real-Time Scheduling Algorithm. *ACM SIGCOM '91*, Zurich.

IEEE 1985a — Institute of Electrical and Electronic Engineers (1985). *Local Area Network – CSMA/CD Access Method and Physical Layer Specifications*. American National Standard ANSI/IEEE 802.3, IEEE Computer Society.

IEEE 1985b — Institute of Electrical and Electronic Engineers (1985). *Local Area Network – Token Bus Access Method and Physical Layer Specifications*. American National Standard ANSI/IEEE 802.4, IEEE Computer Society.

IEEE 1985c — Institute of Electrical and Electronic Engineers (1985). *Local Area Network – Token Ring Access Method and Physical Layer Specifications*. American National Standard ANSI/IEEE 802.5, IEEE Computer Society.

IEEE 1990 — Institute of Electrical and Electronic Engineers (1990). *IEEE Standard 802: Overview and Architecture*. American National Standard ANSI/IEEE 802, IEEE Computer Society.

IEEE 1994 — Institute of Electrical and Electronic Engineers (1994). *Local and Metropolitan Area Networks – Part 6: Distributed Queue Dual Bus (DQDB) Access Method and Physical Layer Specifications.* American National Standard ANSI/IEEE 802.6, IEEE Computer Society.

IEEE 1999 — Institute of Electrical and Electronic Engineers (1999). *Local and Metropolitan Area Networks – Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*. American National Standard ANSI/IEEE 802.11, IEEE Computer Society.

IEEE 2002 — Institute of Electrical and Electronic Engineers (2002). *Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Wireless Personal Area Networks (WPANs).* American National Standard ANSI/IEEE 802.15.1, IEEE Computer Society.

IEEE 2003 — Institute of Electrical and Electronic Engineers (2003). *Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs).* American National Standard ANSI/IEEE 802.15.4, IEEE Computer Society.

IEEE 2004a — Institute of Electrical and Electronic Engineers (2004). *IEEE Standard for Local and Metropolitan Area Networks – Part 16: Air Interface for Fixed Broadband Wireless Access Systems*. American National Standard ANSI/IEEE 802.16, IEEE Computer Society.

IEEE 2004b — Institute of Electrical and Electronic Engineers (2004). *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: Medium Access Control (MAC) Security Enhancement,*. American National Standard ANSI/IEEE 802.11i, IEEE Computer Society.

Imielinski and Navas 1999 — Imielinski, T. and Navas, J.C. (1999). GPS-based geographic addressing, routing, and resource discovery. *Comms. ACM*, Vol. 42, No. 4, pp. 86–92.

International PGP — *The International PGP Home Page*.

Internet World Stats 2004 — Internet World Stats. www.internetworldstats.com

Ishii and Ullmer 1997 — Ishii, H. and Ullmer, B.,(1997). Tangible bits: Towards seamless interfaces between people, bits and atoms. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '97)*, Atlanta, GA, March, pp. 234–241.

ISO 1992 — International Organization for Standardization (1992). *Basic Reference Model of Open Distributed Processing, Part 1: Overview and Guide to Use*. ISO/IEC JTC1/SC212/WG7 CD 10746-1, International Organization for Standardization.

ISO 8879 — International Organization for Standardization (1986). *Information Processing – Text and Office Systems – Standard Generalized Markup Language (SGML)*.

ITU/ISO 1997 — International Telecommunication Union / International Organization for Standardization (1997). Recommendation X.500 (08/97): *Open Systems Interconnection – The Directory: Overview of Concepts, Models and Services*.

Iyer *et al.* 2002 — Iyer, S., Rowstron, A. and Druschel, P. (2002). Squirrel: A decentralized peer-to-peer web cache. In *Proceedings of the 12th ACM Symposium on Principles of Distributed Computing (PODC 2002)*, pp. 213–22.

jakarta.apache.org — The Apache foundation. *Apache Tomcat*.

java.sun.com I — Sun Microsystems. *Java Remote Method Invocation*.

java.sun.com II — Sun Microsystems. *Java Object Serialization Specification*.

java.sun.com III — Sun Microsystems. *Servlet Tutorial*.

java.sun.com IV — Jordan, M. and Atkinson, M. (1999). *Orthogonal Persistence for the Java Platform - Draft Specification*. Sun Microsystems Laboratories, Palo Alto, CA.

java.sun.com V — Sun Microsystems, *Java Security API*.

java.sun.com VI — Sun Microsystems (1999). *JavaSpaces technology*.

java.sun.com VII — Sun Microsystems. *The Java Web Services Tutorial*.

java.sun.com VIII — Sun Microsystems (2003). *Java Data Objects (JDO)*.

java.sun.com IX	Sun Microsystems. *Java Remote Object Activation Tutorial.*
java.sun.com X	Sun Microsystems. *JavaSpaces Service Specification.*
java.sun.com XI	Sun Microsystems. *Java Messaging Service (JMS) home page.*
java.sun.com XII	Sun Microsystems. *Enterprise JavaBeans Specification.*
java.sun.com XIII	Sun Microsystems. *Java Persistence API Specification.*
Johanson and Fox 2004	Johanson, B. and Fox, A. (2004). Extending tuplespaces for coordination in interactive workspaces. *Journal of Systems and Software*, Vol. 69, No. 3, pp. 243–266.
Johnson and Zwaenepoel 1993	Johnson, D. and Zwaenepoel, W. (1993). The peregrine high-performance RPC system. S*oftware–Practice and Experience*, Vol. 23, No. 2, pp. 201–21.
jonas.ow2.org	OW2 Consortium. *JOnAS Application Server*.
Jordan 1996	Jordan, M. (1996). Early experiences with persistent Java. In *Proceedings of the First International Workshop on Persistence and Java*. Glasgow, Scotland, September, pp. 1–9..
Joseph *et al.* 1997	Joseph, A., Tauber, J. and Kaashoek, M. (1997). Mobile Computing with the Rover Toolkit. *IEEE Transactions on Computers: Special Issue on Mobile Computing*, Vol. 46, No. 3, pp. 337–52.
Jul *et al.* 1988	Jul, E., Levy, H., Hutchinson, N. and Black, A. (1988). Fine-grained mobility in the Emerald system. *ACM Transactions on Computer Systems*, Vol. 6, No. 1, pp. 109–33.
jxta.dev.java.net	Jxta Community. *Home page*.
Kaashoek and Tanenbaum 1991	Kaashoek, F. and Tanenbaum, A. (1991). Group communication in the Amoeba distributed operating system. In Proceedings of the 11th International Conference on Distributed Computer Systems, pp. 222–30.
Kaashoek *et al.* 1989	Kaashoek, F., Tanenbaum, A., Flynn Hummel, S. and Bal, H. (1989). An efficient reliable broadcast protocol. *Operating Systems Review*, Vol. 23, No. 4, pp. 5–20.
Kaashoek *et al.* 1997	Kaashoek, M., Engler, D., Ganzer, G., Briceño, H., Hunt, R., Mazières, D., Pinckney, T., Grimm, R., Jannotti, J. and Mackenzie, K. (1997). Application performance and flexibility on exokernel systems. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles*, pp. 52–65.
Kahn 1967	Kahn, D. (1967). *The Codebreakers: The Story of Secret Writing*. New York: Macmillan.
Kahn 1983	Kahn, D. (1983). *Kahn on Codes*. New York: Macmillan.
Kahn 1991	Kahn, D. (1991). *Seizing the Enigma*. Boston: Houghton Mifflin.

Kaler 2002 — Kaler, C. (ed.) (2002). *Specification: Web Services Security (WS-Security)*.

Kaliski and Staddon 1998 — Kaliski, B. and Staddon, J. (1998). *RSA Cryptography Specifications*, Version 2.0. Internet RFC 2437.

Kantor and Lapsley 1986 — Kantor, B. and Lapsley, P. (1986). *Network News Transfer Protocol: A Proposed Standard for the Stream-Based Transmission of News*. Internet RFC 977.

Kehne *et al.* 1992 — Kehne, A., Schonwalder, J. and Langendorfer, H. (1992). A nonce-based protocol for multiple authentications. *ACM Operating Systems Review*, Vol. 26, No. 4, pp. 84–9.

Keith and Wittle 1993 — Keith, B.E. and Wittle, M. (1993). LADDIS: The next generation in NFS file server benchmarking, *USENIX Association Conference Proceedings*, Berkeley, CA, June, pp. 261–78.

Kiciman and Fox 2000 — Kiciman, E. and Fox, A. (2000). Using dynamic mediation to integrate COTS entities in a ubiquitous computing environment. In *Proceedings of the Second International Symposium on Handheld and Ubiquitous Computing (HUC2K)*, Bristol, England, September, pp. 211–226.

Kille 1992 — Kille, S. (1992). *Implementing X.400 and X.500: The PP and QUIPU Systems*. Boston, MA: Artech House.

Kindberg 1995 — Kindberg, T. (1995). A sequencing service for group communication (abstract). In *Proceedings of the 14th Annual ACM Symposium on Principles of Distributed Computing*, p. 260. Technical Report No. 698, Queen Mary and Westfield College Dept. of CS, 1995.

Kindberg 2002 — Kindberg, T. (2002). Implementing physical hyperlinks using ubiquitous identifier resolution. In *Proceedings of the Eleventh International World Wide Web Conference (WWW2002)*, Honolulu, HI, May pp. 191–199.

Kindberg and Barton 2001 — Kindberg, T. and Barton, J. (2001). A web-based nomadic computing system. *Computer Networks*, Vol. 35, No. 4, pp. 443–456.

Kindberg and Fox 2001 — Kindberg, T. and Fox, A. (2001). System software for ubiquitous computing. *IEEE Pervasive Computing*, Vol. 1, No. 1, pp. 70–81.

Kindberg and Zhang 2003a — Kindberg, T. and Zhang, K. (2003). Secure spontaneous device association. In *Proceedings of the Fifth International Conference on Ubiquitous Computing (Ubicomp 2003)*, Seattle, WA, October, pp. 124–131.

Kindberg and Zhang 2003b — Kindberg, T. and Zhang, K. (2003). Validating and securing spontaneous associations between wireless devices. In *Proceedings of the 6th Information Security Conference (ISC'03)*, Bristol, England, October, pp. 44–53.

Kindberg *et al.* 1996 — Kindberg, T., Coulouris, G., Dollimore, J. and Heikkinen, J. (1996). Sharing objects over the Internet: The Mushroom approach. In *Proceedings of the IEEE Global Internet 1996*, London, England, Nov., pp. 67–71.

Kindberg *et al*. 2002a — Kindberg, T., Barton, J., Morgan, J., Becker, G., Bedner, I., Caswell, D., Debaty, P., Gopal, G., Frid, M., Krishnan, V., Morris, H., Pering, C., Schettino, J. and Serra, B. (2002). People, places, things: Web presence for the real world. *Mobile Networks and Applications (MONET)*, Vol. 7, No. 5,, pp. 365–376.

Kindberg *et al.* 2002b — Kindberg, T., Zhang, K. and Shanka, N. (2002). Context authentication using constrained channels. In *Proceedings of the 4th IEEE Workshop on Mobile Computing Systems & Applications (WMCSA 2002)*, Callicoon, NY, June, pp. 14–21.

Kirsch and Amir 2008 — Kirsch, J. and Amir, Y. (2008). Paxos for system builders: An overview. In *Proceedings of the 2nd Workshop on Large-Scale Distributed Systems and Middleware (LADIS '08)*, Vol. 341, Yorktown Heights, NY, pp. 1–6.

Kistler and Satyanarayanan 1992 — Kistler, J.J. and Satyanarayanan, M. (1992). Disconnected operation in the Coda file system. *ACM Transactions on Computer Systems*, Vol. 10, No. 1, pp. 3–25.

Kleinrock 1961 — Kleinrock, L. (1961). *Information Flow in Large Communication Networks*. MIT, RLE Quarterly Progress Report, July.

Kleinrock 1997 — Kleinrock, L. (1997). Nomadicity: Anytime, anywhere in a disconnected world. *Mobile Networks and Applications*, Vol. 1, No. 4, pp. 351–7.

Kohl and Neuman 1993 — Kohl, J. and Neuman, C. (1993). *The Kerberos Network Authentication Service (V5)*. Internet RFC 1510.

Kon *et al*. 2002 — Kon, F., Costa, F., Blair, G. and Campbell, R. (2002). The case for reflective middleware. *Comms. ACM*, Vol. 45, No. 6, pp. 33–38.

Konstantas *et al.* 1997 — Konstantas, D., Orlarey, Y., Gibbs, S. and Carbonel, O. (1997). Distributed music rehearsal. In *Proceedings of the International Computer Music Conferen*ce 97, pp 54–64.

Kopetz and Verissimo 1993 — Kopetz, H. and Verissimo, P. (1993). Real time and dependability concepts. In Mullender, (eds.), *Distributed Systems*, 2nd edn. Reading, MA: Addison-Wesley.

Kopetz *et al.* 1989 — Kopetz, H., Damm, A., Koza, C., Mulazzani, M., Schwabl, W. ,Senft, C. and Zainlinger, R. (1989). Distributed fault-tolerant real-time systems – The MARS Approach. *IEEE Micro*, Vol. 9, No. 1, pp. 112–26.

Krawczyk *et al.* 1997 — Krawczyk, H., Bellare, M. and Canetti, R. (1997). *HMAC: Keyed-Hashing for Message Authentication*. Internet RFC 2104.

Krumm *et al*. 2000 — Krumm, J., Harris, S., Meyers, B., Brumitt, B., Hale, M. and Shafer, S. (2000). Multi-camera multi-person tracking for EasyLiving. In *Proceedings of the Third IEEE International Workshop on Visual Surveillance (VS'2000)*, Dublin, Ireland, July, pp. 3–10.

Kshemkalyani and Singhal 1991 — Kshemkalyani, A. and Singhal, M. (1991). Invariant-based verification of a distributed deadlock detection algorithm. *IEEE Transactions on Software Engineering*, Vol. 17, No. 8, pp. 789–99.

Kshemkalyani and Singhal 1994 — Kshemkalyani, A. and Singhal, M. (1994). On characterisation and correctness of distributed deadlock detection. *Journal of Parallel and Distributed Computing*, Vol. 22, No. 1, pp. 44–59.

Kubiatowicz 2003 — Kubiatowicz, J. (2003). Extracting guarantees from chaos, *Communications of the ACM*, vol. 46, No. 2, pp. 33–38.

Kubiatowicz *et al.* 2000 — Kubiatowicz, J., Bindel, D., Chen, Y., Czerwinski, S., Eaton, P., Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Weimer, W., Wells, C. and Zhao, B. (2000). OceanStore: an architecture for global-scale persistent storage. In *Proc. ASPLOS 2000*, November, pp. 190–201.

Kung and Robinson 1981 — Kung, H.T. and Robinson, J.T. (1981). Optimistic methods for concurrency control. *ACM Transactions on Database Systems*, Vol. 6, No. 2, pp. 213–26.

Kurose and Ross 2007 — Kurose, J.F. and Ross, K.W. (2007). *Computer Networking: A Top-Down Approach Featuring the Internet*. Boston, MA: Addison-Wesley Longman.

Ladin *et al*. 1992 — Ladin, R., Liskov, B., Shrira, L. and Ghemawat, S. (1992). Providing availability using lazy replication. *ACM Transactions on Computer Systems*, Vol. 10, No. 4, pp. 360–91.

Lai 1992 — Lai, X. (1992). On the design and security of block ciphers. *ETH Series in Information Processing*, Vol. 1. Konstanz, Geemany: Hartung-Gorre Verlag.

Lai and Massey 1990 — Lai, X. and Massey, J. (1990). A proposal for a new block encryption standard. In *Proceedings Advances in Cryptology–Eurocrypt '90*, pp. 389–404.

Lamport 1978 — Lamport, L. (1978). Time, clocks and the ordering of events in a distributed system. *Comms. ACM*, Vol. 21, No. 7, pp. 558–65.

Lamport 1979 — Lamport, L. (1979). How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Transactions on Computers*, Vol. C-28, No. 9, pp. 690–1.

Lamport 1986 — Lamport, L. (1986). On interprocess communication, parts I and II. *Distributed Computing*, Vol. 1, No. 2, pp. 77–101.

Lamport 1989 — Lamport, L. (1989). *The Part-Time Parliament*. Technical Report 49, DEC SRC, Palo Alto, CA.

Lamport 1998 — Lamport, L. (1998). The part-time parliament. *ACM Transactions on Computer Systems (TOCS)*, Vol. 16, No. 2, pp. 133–69.

Lamport *et al*. 1982 — Lamport, L., Shostak, R. and Pease, M. (1982). Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, Vol. 4, No. 3, pp. 382–401.

Lampson 1971 — Lampson, B. (1971). Protection. In *Proceedings of the 5th Princeton Conference on Information Sciences and Systems*, p. 437. Reprinted in *ACM Operating Systems Review*. Vol. 8, No. 1, p. 18.

Lampson 1981 — Lampson, B.W. (1981). Atomic transactions. In *Distributed systems: Architecture and Implementation. Vol 105 of Lecture Notes in Computer Science,* Springer-Verlag, pp. 254–9.

Lampson 1986 — Lampson, B.W. (1986). Designing a global name service. In *Proceedings of the 5th ACM Symposium on Principles of Distributed Computing*, pp. 1–10.

Lampson *et al*. 1992 — Lampson, B.W., Abadi, M., Burrows, M. and Wobber, E. (1992). Authentication in distributed systems: Theory and practice. *ACM Transactions on Computer Systems*, Vol. 10, No. 4, pp. 265–310.

Langheinrich 2001 — Langheinrich, M. (2001). Privacy by design – principles of privacy-aware ubiquitous systems. In *Proceedings of the Third International Conference on Ubiquitous Computing (Ubicomp 2001)*, Atlanta, GA, Sep.–Oct., pp. 273–291.

Langville and Meyer 2006 — Lanville, A.M. and Meyer, C.D. (2006). *Pagerank and Beyond: The Science of Search Engine Rankings*. Princeton, NJ: Princeton University Press.

Langworthy 2004 — Langworthy, D. (ed.) (2004) *Web Services Coordination. (WS-Coordination)*, IBM, Microsoft, BEA.

Leach *et al*. 1983 — Leach, P.J., Levine, P.H., Douros, B.P., Hamilton, J.A., Nelson, D.L. and Stumpf, B.L. (1983). The architecture of an integrated local network. *IEEE J. Selected Areas in Communications*, Vol. SAC-1, No. 5, pp. 842–56.

Lee and Thekkath 1996 — Lee, E.K. and Thekkath, C.A. (1996). Petal: Distributed virtual disks, In *Proc. of the 7th Intl. Conf. on Architectural Support for Prog. Langs. and Operating Systems*, October, pp. 84–96.

Lee *et al.* 1996 — Lee, C., Rajkumar, R. and Mercer, C. (1996). Experiences with Processor Reservation and Dynamic QOS in Real-Time Mach. In *Proceedings Multimedia Japan '96.*

Leffler *et al*. 1989 — Leffler, S., McKusick, M., Karels, M. and Quartermain, J. (1989). *The Design and Implementation of the 4.3 BSD UNIX Operating System*. Reading, MA: Addison-Wesley.

Leibowitz *et al.* 2003 — Leibowitz, N., Ripeanu, M. and Wierzbicki, A. (2003). Deconstructing the Kazaa network. In *Proc. of the 3rd IEEE Workshop on Internet Applications (WIAPP'03)*, Santa Clara, CA, p.112.

Leiner *et al.* 1997 — Leiner, B.M., Cerf, V.G., Clark, D.D., Kahn, R.E., Kleinrock, L., Lynch, D.C., Postel, J., Roberts, L.G. and Wolff, S. (1997). A brief history of the Internet. *Comms. ACM*, Vol. 40, No. 1, pp. 102–108.

Leland *et al.* 1993 — Leland, W.E., Taqqu, M.S., Willinger, W. and Wilson, D.V. (1993). On the self-similar nature of Ethernet traffic. *ACM SIGCOMM '93*, San Francisco.

Leslie *et al.* 1996 — Leslie, I., McAuley, D., Black, R., Roscoe, T., Barham, P., Evers, D., Fairbairns, R. and Hyden, E. (1996). The design and implementation of an operating system to support distributed multimedia applications. *ACM Journal of Selected Areas in Communication*, Vol. 14, No. 7, pp. 1280–97.

Liedtke 1996 — Liedtke, J. (1996). Towards real microkernels. *Comms. ACM*, Vol. 39, No. 9, pp. 70–7.

Linux AFS — *The Linux AFS FAQ*.

Liskov 1988 — Liskov, B. (1988). Distributed programming in Argus. *Comms. ACM,* Vol. 31, No. 3, pp. 300–12.

Liskov 1993 — Liskov, B. (1993). Practical uses of synchronized clocks in distributed systems. *Distributed Computing*, Vol. 6, No. 4, pp. 211–19.

Liskov and Scheifler 1982 — Liskov, B. and Scheifler, R.W. (1982). Guardians and actions: Linguistic support for robust, distributed programs. *ACM Transactions on Programming Languages and Systems*, Vol. 5, No. 3, pp. 381–404.

Liskov and Shrira 1988 — Liskov, B. and Shrira, L. (1988). Promises: Linguistic support for efficient asynchronous procedure calls in distributed systems. In *Proceedings of the SIGPLAN '88 Conference Programming Language Design and Implementation*. Atlanta, GA, pp. 260–7.

Liskov *et al*. 1991 — Liskov, B., Ghemawat, S., Gruber, R., Johnson, P., Shrira, L. and Williams, M. (1991). Replication in the Harp file system. In *Proceedings of the 13th ACM Symposium on Operating System Principles*, pp. 226–38.

Liu and Albitz 1998 — Liu, C. and Albitz, P. (1998). *DNS and BIND*, third edition. O'Reilly.

Liu and Layland 1973 — Liu, C.L. and Layland, J.W. (1973). Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, Vol. 20, No. 1, pp. 46–61.

Liu *et al*. 2005 — Liu, J., Sacchetti, D., Sailhan, F. and Issarny, V. (2005). Group management for mobile ad hoc networks: Design, implementation and experiment. In *Proceedings of the 6th international Conference on Mobile Data Management*, New York, pp. 192–199.

Liu *et al*. 2008	Liu, J., Rao, S.G., Li, B. and Zhang, H. (2008). Opportunities and challenges of peer-to-peer Internet video broadcast. In *Proceedings of the IEEE*, *Special Issue on Recent Advances in Distributed Multimedia Communications*, Vol. 96, No. 1, pp. 11–24.
Loepere 1991	Loepere, K. (1991). *Mach 3 Kernel Principles*. Open Software Foundation and Carnegie-Mellon University.
Lundelius and Lynch 1984	Lundelius, J. and Lynch, N. (1984). An upper and lower bound for clock synchronization. *Information and Control*, Vol. 62, No. 2/3, pp. 190–204.
Lv *et al.* 2002	Lv, Q., Cao, P., Cohen, E., Li, K.,and Shenker, S. (2002). Search and replication in unstructured peer-to-peer networks. In *Proceedings of the 2002 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, Marina Del Rey, CA, pp. 258–259.
Lynch 1996	Lynch, N. (1996). *Distributed Algorithms*. San Francisco, CA: Morgan Kaufmann.
Ma 1992	Ma, C. (1992). *Designing a Universal Name Service*. Technical Report 270, University of Cambridge.
Macklem 1994	Macklem, R. (1994). Not quite NFS: Soft cache consistency for NFS. In *Proceedings of the Winter '94 USENIX Conference*, San Francisco, CA, January, pp. 261–278.
Madhavapeddy *et al*. 2003	Madhavapeddy, A., Scott, D. and Sharp, R. (2003). Context-aware computing with sound. In *Proceedings of the Fifth International Conference on Ubiquitous Computing (Ubicomp 2003),* Seattle, WA, October, pp. 315–332.
Maekawa 1985	Maekawa, M. (1985). A $\sqrt{N}$ algorithm for mutual exclusion in decentralized systems. *ACM Transactions on Computer Systems*, Vol. 3, No. 2, pp. 145–159.
Maffeis 1995	Maffeis, S. (1995). Adding group communication and fault tolerance to CORBA. In *Proceedings of the 1995 USENIX Conference on Object-Oriented Technologies*. Monterey, CA. pp. 135–146.
Magee and Sloman 1989	Magee, J. and Sloman, M. (1989). Constructing distributed systems in Conic. *IEEE Trans. Software Engineering* Vol. 15, No. 6, pp. 663–675.
Malkin 1993	Malkin, G. (1993). *RIP Version 2 – Carrying Additional Information*, Internet RFC 1388.
Mamei and Zambonelli 2009	Mamei, M. and Zambonelli, F. (2009). Programming pervasive and mobile computing applications: The TOTA approach. *ACM Transactions on Software Engineering and Methodology*, Vol. 19, No. 4, p. 263.
maps.google.com	Google Maps. *Home page.*

Marsh *et al*. 1991 — Marsh, B., Scott, M., LeBlanc, T. and Markatos, E. (1991). First-class user-level threads. In *Proceedings of the 13th ACM Symposium on Operating System Principles*, pp. 110–21.

Martin *et al*. 2004 — Martin, T., Hsiao, M., Ha, D. and Krishnaswami, J. (2004). Denial-of-service attacks on battery-powered mobile computers. In *Proceedings of the 2nd IEEE Pervasive Computing Conference*, Orlando, FL, March, pp. 309–318.

Marzullo and Neiger 1991 — Marzullo, K. and Neiger, G. (1991). Detection of global state predicates. In *Proceedings of the 5th International Workshop on Distributed Algorithms*, pp. 254–72.

Mattern 1989 — Mattern, F. (1989). Virtual time and global states in distributed systems. In *Proceedings of the Workshop on Parallel and Distributed Algorithm*s, Amsterdam, North-Holland, pp. 215–26.

Maymounkov and Mazieres 2002 — Maymounkov, P. and Mazieres, D. (2002). Kademlia: A peer-to-peer information system based on the XOR metric. In *Proceedings of IPTPS02*, Cambridge, MA, pp. 53–65.

mbone — *BIBs: Introduction to the Multicast Backbone.*

McGraw and Felden 1999 — McGraw, G. and Felden, E. (1999). *Securing Java*. New York: John Wiley & Sons.

McKusick and Quinlan 2010 — McKusick, K. and Quinlan, S. (2010). GFS: Evolution or Fast-Forward. *Comms. ACM*, Vol. 53, No. 3, pp. 42–49.

Meier and Cahill 2010 — Meier, R. and Cahill, V. (2010). On event-based middleware for location-aware mobile applications. *IEEE Transactions on Software Engineering*, Vol. 36, No. 3, pp.09–430.

Melliar-Smith *et al.* 1990 — Melliar-Smith, P., Moser, L. and Agrawala, V. (1990). Broadcast protocols for distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 1, No. 1, pp. 17–25.

Menezes 1993 — Menezes, A. (1993). *Elliptic Curve Public Key Cryptosystems*. Dordrecht, The Netherlands: Kluwer Academic Publishers.

Menezes *et al.* 1997 — Menezes, A., van Oorschot, O. and Vanstone, S. (1997). *Handbook of Applied Cryptography*. Boca Raton, FL: CRC Press.

Metcalfe and Boggs 1976 — Metcalfe, R.M. and Boggs, D.R. (1976). Ethernet: distributed packet switching for local computer networks. *Comms. ACM*, Vol. 19, No. , pp. 395–403.

Milanovic *et al*. 2004 — Milanovic, N., Malek, M., Davidson, A. and Milutinovic, V. (2004). Routing and cecurity in mobile ad hoc networks. *IEEE Computer*, Vol. 37, No. 2, pp. 69–73.

Mills 1995 — Mills, D. (1995). Improved algorithms for synchronizing computer network clocks. *IEEE Transactions Networks*, Vol. 3, No. 3, pp. 245–54.

Milojicic *et al*. 1999	Milojicic, J., Douglis, F. and Wheeler, R. (1999). *Mobility, Processes, Computers and Agents*. Reading, MA: Addison-Wesley.
Mitchell and Dion 1982	Mitchell, J.G. and Dion, J. (1982). A comparison of two network-based file servers. *Comms. ACM*, Vol. 25, No. 4, pp. 233–45.
Mitchell *et al*. 1992	Mitchell, C.J., Piper, F. and Wild, P. (1992). Digital signatures. In Simmons, G.J. (ed.), *Contemporary Cryptology*. New York: IEEE Press.
Mockapetris 1987	Mockapetris, P. (1987). *Domain names – concepts and facilities*. Internet RFC 1034.
Mogul 1994	Mogul, J.D. (1994). Recovery in Spritely NFS. *Computing Systems*, Vol. 7, No. 2, pp. 201–62.
Mok 1985	Mok, A.K. (1985). SARTOR – A design environment for real-time systems. In *Proc. Ninth IEEE COMP-SAC,* Chicago, IL, Octobep, pp. 174–81.
Morin 1997	Morin, R. (ed.) (1997). *MkLinux: Microkernel Linux for the Power Macintosh*. Prime Time Freeware.
Morris *et al.* 1986	Morris, J., Satyanarayanan, M., Conner, M.H., Howard, J.H., Rosenthal, D.S. and Smith, F.D. (1986). Andrew: A distributed personal computing environment. *Comms. ACM*, Vol. 29, No. 3, pp. 184–201.
Moser *et al.* 1994	Moser, L., Amir, Y., Melliar-Smith, P. and Agarwal, D. (1994). Extended virtual synchrony. In *Proceedings of the 14th International Conference on Distributed Computing Systems*, pp. 56–65.
Moser *et al.* 1996	Moser, L., Melliar-Smith, P., Agarwal, D., Budhia, R. and Lingley-Papadopoulos, C. (1996). Totem: A fault-tolerant multicast group communication system. *Comms. ACM*, Vol. 39, No. 4, pp. 54–63.
Moser *et al.* 1998	Moser, L., Melliar-Smith, P. and Narasimhan, P. (1998). Consistent object replication in the Eternal system. *Theory and Practice of Object Systems*, Vol. 4, No. 2, pp. 81–92.
Moss 1985	Moss, E. (1985). *Nested Transactions, An Approach to Reliable Distributed Computing.* Cambridge, MA: MIT Press.
Multimedia Directory	Multimedia Directory (2005). Scala Inc.
Murphy *et al*. 2001	Murphy, A.L., Picco, G.P. and Roman, G.-C. (2001). Lime: A middleware for physical and logical mobility. In *Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS-21)*, Phoenix, AZ, April, pp. 524–233.
Muthitacharoen *et al.* 2002	Muthitacharoen, A., Morris, R., Gil, T.M. and Chen, B. (2002). Ivy: A read/write peer-to-peer file system. In *Proc. Fifth Symposium on Operating Systems Design and Implementation (OSDI)*, Boston, MA, December, pp. 31–44.

Muhl *et al*. 2006 — Muhl, G., Fiege, L. and Pietzuch, P.R. (2006). *Distributed Event-based Systems*. Berlin, Heidelberg: Springer-Verlag.

Myers and Liskov 1997 — Myers, A.C. and Liskov, B. (1997). A decentralized model for information flow control, *ACM Operating Systems Review*, Vol. 31, No. 5, pp. 129–42.

Nagle 1984 — Nagle, J. (1984). Congestion control in TCP/IP internetworks, *Computer Communications Review*, Vol. 14, No. 4, pp. 11–17.

Nagle 1987 — Nagle, J. (1987). On packet switches with infinite storage. *IEEE Transactions on Communications*, Vol. 35, No. 4, pp. 435–8.

National Bureau of Standards 1977 — National Bureau of Standards (1977). *Data Encryption Standard (DES).* Federal Information Processing Standards No. 46, Washington, DC: US National Bureau of Standards.

nbcr.sdsc.edu — National Biomedical Computation Resource, University of California, San Diego.

Needham 1993 — Needham, R. (1993). Names. In Mullender, S. (ed.), *Distributed Systems*, *an Advanced Course*, 2nd edn. Wokingham, England: ACM Press/Addison-Wesley. pp. 315–26.

Needham and Schroeder 1978 — Needham, R.M. and Schroeder, M.D. (1978). Using encryption for authentication in large networks of computers. *Comms. ACM*, Vol. 21, No. 12, pp. 993–9.

Nelson *et al.* 1988 — Nelson, M.N., Welch, B.B. and Ousterhout, J.K. (1988). Caching in the Sprite network file system. *ACM Transactions on Computer Systems*, Vol. 6, No. 1, pp. 134–154.

Neuman *et al.* 1999 — Neuman, B.C., Tung, B. and Wray, J. (1999). *Public Key Cryptography for Initial Authentication in Kerberos*. Internet Draft ietf-cat-kerberos-pk-init-09.

Neumann and Ts'o 1994 — Neuman, B.C. and Ts'o, T. (1994). Kerberos: An authentication service for computer networks. *IEEE Communications*, Vol. 32, No. 9, pp. 33–38.

Newcomer 2002 — Newcomer, E. (2002). *Understanding Web Services XML, WSDL, SOAP and UDDI*. Boston, MA: Pearson.

Nielsen and Thatte 2001 — Nielsen, H.F. and Thatte, S. (2001). *Web Services Routing Protocol (WS-Routing)*. Microsoft Corporation.

Nielsen *et al.* 1997 — Nielsen, H., Gettys, J., Baird-Smith, A., Prud'hommeaux, E., Lie, H. and Lilley, C. (1997). Network performance effects of HTTP/1.1, CSS1, and PNG. In *Proceedings of the SIGCOMM '97*, pp. 155–66.

NIST 1994 — National Institute for Standards and Technology (1994). *Digital Signature Standard*, NIST FIPS 186. US Department of Commerce.

NIST 2002 — National Institute for Standards and Technology (2002). *Secure Hash Standard.* NIST FIPS 180-2 + Change Notice to include SHA-224. US Department of Commerce.

NIST 2004 — National Institute for Standards and Technology (2004). *NIST Brief Comments on Recent Cryptanalytic Attacks on Secure Hashing Functions and the Continued Security Provided by SHA-1*. US Department of Commerce.

nms.csail.mit.edu — The Berkeley RON project. *Home page*.

Noble and Satyanarayanan 1999 — Noble, B. and Satyanarayanan, M. (1999). Experience with adaptive mobile applications in Odyssey. *Mobile Networks and Applications*, Vol. 4 , No. 4, pp. 245–254.

now.cs.berkeley.edu — The Berkeley NOW project. *Home page*.

Oaks and Wong 1999 — Oaks, S. and Wong, H. (1999). *Java Threads*, 2nd edn. Sebastoplo, CA: O'Reilly.

Ohkubo *et al*. 2003 — Ohkubo, M., Suzuki, K. and Kinoshita, S. (2003). Cryptographic approach to 'privacy-friendly' tags. In *Proceedings of the RFID Privacy Workshop*, MIT.

Oki *et al*. 1993 — Oki, B., Pfluegl, M., Siegel, A., and Skeen, D. (1993). The Information Bus: an architecture for extensible distributed systems. In *Proceedings of the Fourteenth ACM Symposium on Operating Systems Principles*, Asheville, NC, December, NY. pp. 58–68.

Olson and Ogbuji 2002 — Olson, M. and Ogbuji, U. (2002). *The Python Web services developer: Messaging technologies compared – Choose the best tool for the task at hand.* IBM DeveloperWorks

OMG 2000a — Object Management Group (2000). *Trading Object Service Specification*, Vn. 1.0. Needham, MA: OMG.

OMG 2000b — Object Management Group (2000). *Concurrency Control Service Specification*. Needham, MA: OMG.

OMG 2002a — Object Management Group (2002). *The CORBA IDL Specification*. Needham, MA: OMG.

OMG 2002b — Object Management Group (2002). *CORBA Security Service Specification* Vn. 1.8. Needham, MA: OMG.

OMG 2002c — Object Management Group (2002). *Value Type Semantics*. Needham, MA: OMG.

OMG 2002d — Object Management Group (2002). *Life Cycle Service*, Vn. 1.2. Needham, MA: OMG.

OMG 2002e — Object Management Group (2002). *Persistent State Service*, Vn. 2.0. Needham, MA: OMG.

OMG 2003 — Object Management Group, (2003). *Object Transaction Service Specification,* Vn. 1.4. Needham, MA: OMG.

OMG 2004a — Object Management Group (2004). *CORBA/IIOP 3.0.3 Specification*. Needham, MA: OMG.

OMG 2004b — Object Management Group (2004). *Naming Service Specification*. Needham, MA: OMG.

OMG 2004c — Object Management Group (2004). *Event Service Specification*, Vn. 1.2. Needham, MA: OMG.

OMG 2004d — Object Management Group (2004). *Notification Service Specification*, Vn. 1.1. Needham, MA: OMG. Technical report telecom/98-06-15.

OMG 2004e — Object Management Group (2004). *CORBA Messaging*. Needham, MA: OMG.

Omidyar and Aldridge 1993 — Omidyar, C.G. and Aldridge, A. (1993). Introduction to SDH/SONET. *IEEE Communications Magazine*, Vol. 31, pp. 30–3.

open.eucalyptus.com — Eucalyptus. *Home page*.

OpenNap 2001 — OpenNap: Open Source Napster Server, Beta release 0.44, September 2001.

Oppen and Dalal 1983 — Oppen, D.C. and Dalal Y.K. (1983). The Clearinghouse: a decentralized agent for locating named objects in a distributed environment. *ACM Trans. on Office Systems*, Vol. 1, No. 3, pp. 230–53.

Oram 2001 — Oram, A. (2001). *Peer-to-Peer: Harnessing the Benefits of Disruptive Technologies*, O'Reilly, Sebastapol, CA.

Orfali *et al*. 1996 — Orfali, R., Harkey, D. and Edwards, J. (1996). *The Essential Distributed Objects Survival Guide*. New York: Wiley.

Orfali *et al*. 1997 — Orfali, R., Harkey, D., and Edwards, J. (1997) *Instant CORBA*. New York: John Wiley & Sons.

Organick 1972 — Organick, E.I. (1972). *The MULTICS System: An Examination of its Structure*. Cambridge, MA: MIT Press.

Orman *et al*. 1993 — Orman, H., Menze, E., O'Malley, S. and Peterson, L. (1993). A fast and general implementation of Mach IPC in a Network. In *Proceedings of the Third USENIX Mach Conference*, April.

OSF — *Introduction to OSF DCE*. The Open Group.

Ousterhout *et al.* 1985 — Ousterhout, J., Da Costa, H., Harrison, D., Kunze, J., Kupfer, M. and Thompson, J. (1985). A Trace-driven analysis of the UNIX 4.2 BSD file system. In *Proc. of the 10th ACM Symposium Operating System Principles, p. 15–24*.

Ousterhout *et al*. 1988 — Ousterhout, J., Cherenson, A., Douglis, F., Nelson, M. and Welch, B. (1988). The Sprite network operating system. *IEEE Computer,* Vol. 21, No. 2, pp. 23–36.

Parker 1992 — Parker, B. (1992). *The PPP AppleTalk Control Protocol (ATCP)*. Internet RFC 1378.

Parrington *et al.* 1995 — Parrington, G.D., Shrivastava, S.K., Wheater, S.M. and Little, M.C. (1995). The design and implementation of Arjuna. *USENIX Computing Systems Journal*, Vol. 8, No. 3, pp. 255–308.

Partridge 1992 — Partridge, C. (1992). *A Proposed Flow Specificatio*n. Internet RFC 1363.

Patel and Abowd 2003 — Patel, S.N. and Abowd, G.D. (2003). A 2-way laser-assisted selection scheme for handhelds in a physical environment. In *Proceedings of the Fifth International Conference on Ubiquitous Computing (Ubicomp 2003),* Seattle, WA, October, pp. 200–207.

Patterson *et al.* 1988 — Patterson, D., Gibson, G. and Katz, R. (1988). A case for redundant arrays of interactive disks. ACM *International Conf. on Management of Data (SIGMOD)*, pp. 109–116.

Pease *et al*. 1980 — Pease, M., Shostak, R. and Lamport, L. (1980). Reaching agreement in the presence of faults. *Journal of the ACM*, Vol. 27, No. 2, pp. 228–34.

Pedone and Schiper 1999 — Pedone, F. and Schiper, A. (1999). Generic broadcast. In *Proceedings of the 13th International Symposium on Distributed Computing* (DISC '99), September, pp. 94–108.

Peng and Dabek 2010 — Peng, D. and Dabex, F. (2010). Large-scale incremental processing using distributed transactions and notifications. In *Proceedings of the Ninth Symposium on Operating Systems Design and Implementation (OSDI '10)*, Vancouver, Canada, October, pp. 1–15.

Perrig *et al*. 2002 — Perrig, A., Szewczyk, R., Wen, V., Culler, D. and Tygar, D. (2002). SPINS: Security protocols for sensor networks. *Wireless Networks*, Vol. 8, No. 5, pp. 521–534.

Peterson 1988 — Peterson, L. (1988). The Profile Naming Service. *ACM Transactions on Computer Systems*, Vol. 6, No. 4, pp. 341–64.

Peterson *et al*. 1989 — Peterson, L.L., Buchholz, N.C. and Schlichting, R.D. (1989). Preserving and using context information in interprocess communication. *ACM Transactions on Computer Systems*, Vol. 7, No. 3, pp. 217–46.

Petersen *et al.* 1997 — Petersen, K., Spreitzer, M., Terry, D., Theimer, M. and Demers, A. (1997). Flexible update propagation for weakly consistent replication. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles*, pp. 288–301.

Peterson *et al*. 2005 — Peterson, L.L., Shenker, S. and Turner, J. (2005). Overcoming the Internet impasse through virtualization. *Computer*, Vol. 38, No. 4, pp. 34–41.

Pietzuch and Bacon 2002 — Pietzuch, P. R. and Bacon, J. (2002). Hermes: A distributed event-based middleware architecture. In *Proceedings of the First International Workshop on Distributed Event-Based Systems*, Vienna, Austria, pp.611–618.

Pike *et al.* 1993 — Pike, R., Presotto, D., Thompson, K., Trickey, H. and Winterbottom, P. (1993). The use of name spaces in Plan 9. *Operating Systems Review*, Vol. 27, No. 2, pp. 72–76.

Pike *et al.* 2005 — Pike, R., Dorward, S., Griesemer, R. and Quinlan, S. (2005). Interpreting the data: Parallel analysis with Sawzall. *Scientific Programming*, Vol. 13, No. 4, pp. 277–298.

Pinheiro *et al.* 2007 — Pinheiro, E., Weber, W.D. and Barroso, L.A. (2007). Failure trends in a large disk drive population. In *Proceedings of the 5th USENIX Conference on File and Storage Technologies*, pp. 17–28.

Plaxton *et al.* 1997 — Plaxton, C.G., Rajaraman, R. and Richa, A.W. (1997). Accessing nearby copies of replicated objects in a distributed environment. In *Proc. of the ACM Symposium on Parallel Algorithms and Architectures*, pp. 311–320.

Ponnekanti and Fox 2004 — Ponnekanti, S. and Fox, A. (2004). Interoperability among independently evolving web services. In *Proceedings of the ACM/Usenix/IFIP 5th International Middleware Conference*, Toronto, Canada, pp. 331–57.

Ponnekanti *et al*. 2001 — Ponnekanti, S.R., Lee, B., Fox, A., Hanrahan, P. and Winograd, T. (2001). ICrafter: A service framework for ubiquitous computing environments. In *Proceedings of the Third International Conference on Ubiquitous Computing (Ubicomp 2001)*, Atlanta, GA, Sep.–Oct., pp. 56–75.

Popek and Goldberg 1974 — Popek, G.J. and Goldberg, R.P. (1974). Formal requirements for virtualizable third generation architectures. Comms. ACM, Vol. 17, No. 7, pp. 412–421.

Popek and Walker 1985 — Popek, G. and Walker, B. (eds.) (1985). *The LOCUS Distributed System Architecture*. Cambridge, MA: MIT Press.

Postel 1981a — Postel, J. (1981). *Internet Protocol.* Internet RFC 791.

Postel 1981b — Postel, J. (1981). *Transmission Control Protocol.* Internet RFC 793.

Pottie and Kaiser 2000 — Pottie, G.J. and Kaiser, W.J. (2000). Embedding the Internet: Wireless integrated network sensors. *Comms. ACM*, Vol. 43, No. 5, pp. 51–58.

Powell 1991 — Powell, D. (ed.) (1991). *Delta-4: A Generic Architecture for Dependable Distributed Computing*. Berlin and New York: Springer-Verlag.

Pradhan and Chiueh 1998 — Pradhan, P. and Chiueh, T. (1998). Real-time performance guarantees over wired and wireless LANS. In *IEEE Conference on Real-Time Applications and Systems*, RTAS'98, June, p. 29.

Prakash and Baldoni 1998 — Prakash, R. and Baldoni, R. (1998). Architecture for group communication in mobile systems. In *Proceedings of the the 17th IEEE Symposium on Reliable Distributed Systems*, Washington, DC, pp. 235–242.

Preneel *et al.* 1998 — Preneel, B., Rijmen, V. and Bosselaers, A. (1998). Recent developments in the design of conventional cryptographic algorithms. In *Computer Security and Industrial Cryptography, State of the Art and Evolution*. Vol. 1528 of Lecture Notes in Computer Science, Springer-Verlag, pp. 106–131.

privacy.nb.ca — *International Cryptography Freedom*.

Radia *et al.* 1993 — Radia, S., Nelson, M. and Powell, M. (1993). *The Spring Naming Service*. Technical Report 93–16, Sun Microsystems Laboratories, Inc.

Raman and McCanne 1999 — Raman, S. and McCanne, S. (1999). A model, analysis, and protocol framework for soft state-based communication. In *Proceedings of the ACM SIGCOMM*, 1999, Cambridge, MA, pp. 15–25.

Randall and Szydlo 2004 — Randall, J. and Szydlo, M. (2004). Collisions for SHA0, MD5, HAVAL, MD4, and RIPEMD, but SHA1 still secure. *RSA Laboratories Technical Note*, August 31.

Rashid 1985 — Rashid, R.F. (1985). Network operating systems. In *Local Area Networks: An Advanced Course, Lecture Notes in Computer Science*, 184, Springer-Verlag, pp. 314–40.

Rashid 1986 — Rashid, R.F. (1986). From RIG to Accent to Mach: the evolution of a network operating system. In *Proceedings of the ACM/IEEE Computer Society Fall Joint Conference*, ACM, November.

Rashid and Robertson 1981 — Rashid, R. and Robertson, G. (1981). Accent: a communications oriented network operating system kernel. *ACM Operating Systems Review*, Vol. 15, No. 5, pp. 64–75.

Rashid *et al*. 1988 — Rashid, R., Tevanian Jr, A., Young, M., Golub, D., Baron, R., Black, D., Bolosky, W.J. and Chew, J. (1988). Machine-Independent Virtual Memory Management for Paged Uniprocessor and Multiprocessor Architectures. *IEEE Transactions on Computers*, Vol. 37, No. 8, pp. 896–907.

Ratnasamy *et al.* 2001 — Ratnasamy, S., Francis, P., Handley, M., Karp, R. and Shenker, S. (2001). A scalable content-addressable network. In *Proc. ACM SIGCOMM 2001*, August, pp. 161–72.

Raynal 1988 — Raynal, M. (1988). *Distributed Algorithms and Protocols*. New York: John Wiley & Sons.

Raynal 1992 — Raynal, M. (1992). About logical clocks for distributed systems. *ACM Operating Systems Review*, Vol. 26, No. 1, pp. 41–8.

Raynal and Singhal 1996 — Raynal, M. and Singhal, M. (1996). Logical time: Capturing causality in distributed systems. *IEEE Computer*, Vol. 29, No. 2, pp. 49–56.

Redmond 1997 — Redmond, F.E. (1997). *DCOM: Microsoft Distributed Component Model*. IDG Books Worldwide.

Reed 1983 — Reed, D.P. (1983). Implementing atomic actions on decentralized data. *ACM Transactions on Computer Systems*, Vol. 1, No. 1, pp. 3–23.

Rellermeyer *et al*. 2007 — Rellermeyer, J. S., Alonso, G., and Roscoe, T. (2007). R-OSGi: Distributed applications through software modularization. In *Proceedings of the ACM/IFIP/USENIX 2007 international Conference on Middleware*, Newport Beach, CA, November, pp. 1–20.

Rescorla 1999 — Rescorla, E. (1999). *Diffie-Hellman Key Agreement Method*. Internet RFC 2631.

research.microsoft.com — Microsoft Research. *Writings of Leslie Lamport*.

Rether — Rether: A Real-Time Ethernet Protocol.

Rhea *et al*. 2001 — Rhea, S., Wells, C., Eaton, P., Geels, D., Zhao, B., Weatherspoon, H. and Kubiatowicz, J. (2001). Maintenance-free global data storage. *IEEE Internet Computing*, Vol. 5, No. 5, pp. 40–49.

Rhea *et al*. 2003 — Rhea, S., Eaton, P., Geels, D., Weatherspoon, H., Zhao, B. and Kubiatowicz, J. (2003). Pond: The OceanStore prototype, In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST '03),* pp. 1–14.

Ricart and Agrawala 1981 — Ricart, G. and Agrawala, A.K. (1981). An optimal algorithm for mutual exclusion in computer networks. *Comms. ACM*, Vol. 24, No. 1, pp. 9–17.

Richardson *et al.* 1998 — Richardson, T., Stafford-Fraser, Q., Wood, K.R. and Hopper, A. (1998). Virtual network computing, *IEEE Internet Computing*, Vol. 2, No. 1, pp. 33–8.

Ritchie 1984 — Ritchie, D. (1984). A Stream Input Output System. *AT&T Bell Laboratories Technical Journal*, Vol. 63, No. 8, pt 2, pp. 1897–910.

Rivest 1992a — Rivest, R. (1992). *The MD5 Message-Digest Algorithm*. Internet RFC 1321.

Rivest 1992b — Rivest, R. (1992). *The RC4 Encryption Algorithm*. RSA Data Security Inc.

Rivest *et al*. 1978 — Rivest, R.L., Shamir, A. and Adelman, L. (1978). A method of obtaining digital signatures and public key cryptosystems. *Comms. ACM*, Vol. 21, No. 2, pp. 120–6.

Rodrigues *et al.* 1998 — Rodrigues, L., Guerraoui, R. and Schiper, A. (1998). Scalable atomic multicast. In *Proceedings IEEE IC3N '98.* Technical Report 98/257. École polytechnique fédérale de Lausanne.

Roman *et al*. 2001 — Roman, G., Huang, Q. and Hazemi, A. (2001). Consistent group membership in ad hoc networks. In *Proceedings of the 23rd International Conference on Software Engineering*, Washington, DC, pp. 381–388.

Rose 1992 — Rose, M.T. (1992). *The Little Black Book: Mail Bonding with OSI Directory Services*. Englewood Cliffs, NJ: Prentice-Hall.

Rosenblum and Ousterhout 1992 — Rosenblum, M. and Ousterhout, J. (1992). The design and implementation of a log-structured file system. *ACM Transactions on Computer Systems*, Vol. 10, No. 1, pp. 26–52.

Rosenblum and Wolf 1997 — Rosenblum, D.S. and Wolf, A.L. (1997). A design framework for Internet-scale event observation and notification. In *Proceedings of the sixth European Software Engineering Conference/ACM SIGSOFT Fifth Symposium on the Foundations of Software Engineering*, Zurich, Switzerland, pp. 344–60.

Rowley 1998 — Rowley, A. (1998). *A Security Architecture for Groupware*. Doctoral Thesis, Queen Mary and Westfield College, University of London.

Rowstron and Druschel 2001 — Rowstron, A. and Druschel, P. (2001). Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proc. IFIP/ACM Middleware 2001*, Heidelberg, Germany, Nov., pp. 329–50.

Rowstron and Wood 1996 — Rowstron, A. and Wood, A. (1996). An efficient distributed tuple space implementation for networks of workstations. In *Proceedings of the Second International Euro-Par Conference*, Lyon, France, pp. 510–513.

royal.pingdom.com — Royal Pingdom. *Map of all Google data centre locations as of 2008*.

Rozier *et al*. 1988 — Rozier, M., Abrossimov, V., Armand, F., Boule, I., Gien, M., Guillemont, M., H6rrman, F., Kaiser, C., Langlois, S., Leonard, P. and Neuhauser, W. (1988). Chorus distributed operating systems. *Computing Systems Journal*, Vol. 1, No. 4, pp. 305–70.

Rozier *et al*. 1990 — Rozier, M., Abrossimov, V., Armand, F., Boule, I., Gien, M., Guillemont, M., Herrman, F., Kaiser, C., Langlois, S., Leonard, P. and Neuhauser, W. (1990). *Overview of the Chorus Distributed Operating System*. Technical Report CS/TR-90-25.1, Chorus Systèmes, France.

RTnet — RTnet: Hard Real-Time Networking for Linux/RTAI.

Salber *et al*. 1999 — Salber, D., Dey, A.K. and Abowd, G.D. (1999). The Context Toolkit: Aiding the development of context-enabled applications. In *Proceedings of the 1999 Conference on Human Factors in Computing Systems (CHI '99)*, Pittsburgh, PA, May, pp. 434–441.

Saltzer *et al.* 1984 — Saltzer, J.H., Reed, D.P. and Clarke, D. (1984). End-to-end arguments in system design. *ACM Transactions on Computer Systems*, Vol. 2, No. 4, pp. 277–88.

Sandberg 1987 — Sandberg, R. (1987). *The Sun Network File System: Design, Implementation and Experience*. Technical Report. Mountain View, CA: Sun Microsystems.

Sandberg *et al*. 1985 — Sandberg, R., Goldberg, D., Kleiman, S., Walsh, D. and Lyon, B. (1985). The design and implementation of the Sun Network File System. In *Proceedings of the Usenix Conference*, Portland, OR, p. 119.

Sanders 1987 — Sanders, B. (1987). The information structure of distributed mutual exclusion algorithms. *ACM Transactions on Computer Systems*, Vol. 5, No. 3, pp. 284–99.

Sandholm and Gawor 2003 — Sandholm, T. and Gawor, J. (2003). *Globus Toolkit 3 Core – A Grid Service Container Framework*. July.

Sandhu *et al.* 1996 — Sandhu, R., Coyne, E., Felstein, H. and Youman, C. (1996). Role-based access control models. *IEEE Computer*, Vol. 29, No. 2, pp. 38–47.

Sansom *et al*. 1986 — Sansom, R.D., Julin, D.P. and Rashid, R.F. (1986). *Extending a capability based system into a network environment*. Technical Report CMU-CS-86-116, Carnegie-Mellon University.

Santifaller 1991 — Santifaller, M. (1991). *TCP/IP and NFS, Internetworking in a Unix Environment*. Reading, MA: Addison-Wesley.

Saroiu *et al.* 2002 — Saroiu, S., Gummadi, P. and Gribble, S. (2002). A measurement study of peer-to-peer file sharing systems. In *Proc. Multimedia Computing and Networking (MMCN)*, pp. 156–70.

Sastry *et al*. 2003 — Sastry, N., Shankar, U. and Wagner, D. (2003). Secure verification of location claims. In *Proceedings of the ACM Workshop on Wireless Security (WiSe 2003)*, September, pp. 1–10.

Satyanarayanan 1981 — Satyanarayanan, M. (1981). A study of file sizes and functional lifetimes. In *Proceedings of the 8th ACM Symposium on Operating System Principles*, Asilomar, CA, pp. 96–108.

Satyanarayanan 1989a — Satyanarayanan, M. (1989). Distributed File Systems. In Mullender, S. (ed.), *Distributed Systems*, *an Advanced Course*, 2nd edn. Wokingham, England: ACM Press/Addison-Wesley. pp. 353–83.

Satyanarayanan 1989b — Satyanarayanan, M. (1989). Integrating security in a large distributed system. *ACM Transactions on Computer Systems*, Vol. 7, No. 3, pp. 247–80.

Satyanarayanan 2001 — Satyanarayanan, M. (2001). Pervasive computing: Vision and challenges. *IEEE Personal Communications*, Vol. 8, No. 4, pp. 10–17.

Satyanarayanan *et al*. 1990 — Satyanarayanan, M., Kistler, J.J., Kumar, P., Okasaki, M.E., Siegel, E.H. and Steere, D.C. (1990). Coda: A highly available file system for a distributed workstation environment. *IEEE Transactions on Computers*, Vol. 39, No. 4, pp. 447–59.

Schilit *et al*. 1994 — Schilit, B.N., Adams, N.I. and Want, R. (1994). Context-aware computing applications. In *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, December, pp. 85–90.

Schiper and Raynal 1996 — Schiper, A. and Raynal, M. (1996). From group communication to transactions in distributed systems. *Comms. ACM*, Vol. 39, No. 4, pp. 84–7.

Schiper and Sandoz 1993 — Schiper, A. and Sandoz, A. (1993). Uniform reliable multicast in a virtually synchronous environment. In *Proceedings of the 13th International Conference on Distributed Computing Systems*, pp. 561–8.

Schlageter 1982 — Schlageter, G. (1982). Problems of optimistic concurrency control in distributed database systems. *SigMOD Record,* Vol. 13, No. 3, pp. 62–6.

Schmidt 1998 — Schmidt, D. (1998). Evaluating architectures for multithreaded object request brokers. *Comms. ACM*, Vol. 44, No. 10, pp. 54–60.

Schneider 1990 — Schneider, F.B. (1990). Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys*, Vol. 22, No. 4, pp. 300–19.

Schneider 1996 — Schneider, S. (1996). Security properties and CSP. In *IEEE Symposium, on Security and Privacy*, pp. 174–187.

Schneier 1996 — Schneier, B. (1996). *Applied Cryptography*, 2nd edn. New York: John Wiley & Sons.

Schroeder and Burrows 1990 — Schroeder, M. and Burrows, M. (1990). The performance of Firefly RPC. *ACM Transactions on Computer Systems*, Vol. 8, No. ,. pp. 1–17.

Schroeder *et al.* 1984 — Schroeder, M.D., Birrell, A.D. and Needham, R.M. (1984). Experience with Grapevine: The growth of a distributed system, *ACM Transactions on Computer Systems*, Vol. 2, No. 1, pp. 3–23.

Schulzrinne *et al.* 1996 — Schulzrinne, H., Casner, S., Frederick, D. and Jacobson, V. (1996). *RTP: A Transport Protocol for Real-Time Applications*. Internet RFC 1889.

sector.sourceforge.net — Sector/Sphere. *Home page.*

Seetharamanan 1998 — Seetharamanan, K. (ed.) (1998). Special Issue: The CORBA Connection. *Comms. ACM*, Vol. 41, No. 10.

session directory — *User Guide to sd (Session Directory)*.

Shannon 1949 — Shannon, C.E. (1949). Communication theory of secrecy systems. *Bell System Technical Journal*, Vol. 28, No. 4, pp. 656–715.

Shepler 1999 — Shepler, S. (1999). *NFS Version 4 Design Considerations*. Internet RFC 2624, Sun Microsystems.

Shih *et al*. 2002 — Shih, E., Bahl, P. and Sinclair, M. (2002). Wake on Wireless: An event driven energy saving strategy for battery operated devices. In *Proceedings of the Eighth Annual ACM Conference on Mobile Computing and Networking*, Altanta, GA, September, pp. 160–171.

Shirky 2000 — Shirky, C. (2000). What's P2P and what's not, 11/24/2000. Internet publication.

Shoch and Hupp 1980 — Shoch, J.F. and Hupp, J.A. (1980). Measured performance of an Ethernet local network. *Comms. ACM*, Vol. 23, No. 12, pp. 711–21.

Shoch and Hupp 1982 — Shoch, J.F. and Hupp, J.A. (1982). The 'Worm' programs – early experience with a distributed computation. *Comms. ACM*, Vol. 25, No. 3, pp. 172–80.

Shoch *et al*. 1982 — Shoch, J.F., Dalal, Y.K. and Redell, D.D. (1982). The evolution of the Ethernet local area network. *IEEE Computer*, Vol. 15, No. 8, pp. 10–28.

Shoch *et al*. 1985 — Shoch, J.F., Dalal, Y.K., Redell, D.D. and Crane, R.C. (1985). The Ethernet. In *Local Area Networks: An Advanced Course. Vol 184 of Lecture Notes in Computer Science.* Springer-Verlag, pp. 1–33.

Shrivastava *et al*. 1991 — Shrivastava, S., Dixon, G.N. and Parrington, G.D. (1991). An overview of the Arjuna distributed programming system. *IEEE Software*, pp. 66–73.

Singh 1999 — Singh, S. (1999). *The Code Book*. London: Fourth Estate.

Sinha and Natarajan 1985 — Sinha, M. and Natarajan, N. (1985). A priority based distributed deadlock detection algorithm. *IEEE Transactions on Software Engineering.* Vol. 11, No. 1, pp. 67–80.

Sirivianos *et al.* 2007 — Sirivianos, M., Park, J.H., Chen R. and Yang, X. (2007). Free-riding in BitTorrent networks with the large view exploit. In *Proceedings of the 6th International Workshop on Peer-to-Peer Systems (IPTPS '07)*, Bellevue, WA.

Spafford 1989 — Spafford, E.H. (1989). The Internet worm: Crisis and aftermath. *Comms. ACM*, Vol. 32, No. 6, pp. 678–87.

Spasojevic and Satyanarayanan 1996 — Spasojevic, M. and Satyanarayanan, M. (1996). An empirical study of a wide-area distributed file system. *ACM Transactions on Computer Systems*, Vol. 14, No. 2, pp. 200–222.

Spector 1982 — Spector, A.Z. (1982). Performing remote operations efficiently on a local computer network. *Comms. ACM*, Vol. 25, No. 4, pp. 246–60.

Spurgeon 2000 — Spurgeon, C.E. (2000). *Ethernet: The Definitive Guide*. Sebastopol, CA: O'Reilly.

Srikanth and Toueg 1987 — Srikanth, T. and Toueg, S. (1987). Optimal clock synchronization. *Journal ACM.* Vol. 34, No. 3, pp. 626–45.

Srinivasan 1995a — Srinivasan, R. (1995). RPC: *Remote Procedure Call Protocol Specification Version 2*. Sun Microsystems. Internet RFC 1831. August.

Srinivasan 1995b — Srinivasan, R. (1995). *XDR: External Data Representation Standard*. Sun Microsystems. Internet RFC 1832. Network Working Group.

Srinivasan and Mogul 1989 — Srinivasan, R. and Mogul, J.D. (1989). Spritely NFS: Experiments with cache-consistency protocols. *In Proc. of the 12th ACM Symposium on Operating System Principles*, Litchfield Park, AZ, December, pp. 45–57.

Srisuresh and Holdrege 1999	Srisuresh, P. and Holdrege, M. (1999). *IP Network Address Translator (NAT) Terminology and Considerations*. Internet RFC 2663.
Stajano 2002	Stajano, F. (2002). *Security for Ubiquitous Computing*. New York: John Wiley & Sons.
Stajano and Anderson 1999	Stajano, F. and Anderson, R. (1999). The resurrecting duckling: Security issues for adhoc wireless networks. In *Proceedings of the 7th International Workshop on Security Protocols*, pp. 172–194.
Stallings 2002	Stallings, W. (2002). *High Speed Networks – TCP/IP and ATM Design Principles*. 2nd edn.Upper Saddle River, NJ: Prentice-Hall.
Stallings 2005	Stallings, W. (2005). *Cryptography and Network Security – Principles and Practice*, 4th edn. Upper Saddle River, NJ: Prentice-Hall.
Stallings 2008	Stallings, W. (2008). *Operating Systems*, 6th edn. Upper Saddle River, NJ: Prentice-Hall International.
Steiner *et al*. 1988	Steiner, J., Neuman, C. and Schiller, J. (1988). Kerberos: An authentication service for open network systems. In *Proceedings of the Usenix Winter Conference*, Berkeley, CA.
Stelling *et al.* 1998	Stelling, P., Foster, I., Kesselman, C., Lee, C. and von Laszewski, G. (1998). A fault detection service for wide area distributed computations. In *Proceedings of the 7th IEEE Symposium on High Performance Distributed Computing*, pp. 268–78.
Stoica *et al*. 2001	Stoica, I., Morris, R., Karger, D., Kaashoek, F. and Balakrishnan, H. (2001). Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. ACM SIGCOMM, '01*, August, pp. 149–60.
Stojmenovic 2002	Stojmenovic, I. (ed.) (2002). *Handbook of Wireless Networks and Mobile Computing*. New York: John Wiley & Sons.
Stoll 1989	Stoll, C. (1989). *The Cuckoo's Egg: Tracking a Spy Through a Maze of Computer Espionage*. New York: Doubleday.
Stone 1993	Stone, H. (1993). *High-performance Computer Architecture*, 3rd edn. Reading, MA: Addison-Wesley.
Stonebraker *et al*. 2010	Stonebraker, M., Abadi, D., DeWitt, D. J., Madden, S., Paulson, E., Pavlo, A. and Rasin, A. (2010). MapReduce and parallel DBMSs: Friends or foes? *Comms. ACM*, Vol. 53, No. 1, pp. 64–71.
Sun 1989	Sun Microsystems Inc. (1989). *NFS: Network File System Protocol Specification*. Internet RFC 1094.
Sun and Ellis 1998	Sun, C. and Ellis, C. (1998). Operational transformation in real-time group editors: Issues, algorithms, and achievements. In *Proceedings of the Conference on Computer Supported Cooperative Work Systems*, pp. 59–68.

SWAP-CA 2002 — Shared Wireless Access Protocol (Cordless Access) Specification (SWAP-CA), Revision 2.0,1. The HomeRF Technical Committee, July 2002.

Szalay and Gray 2001 — Szalay, A. and Gray, J. (2001) The World-Wide Telescope. *Science*, Vol. 293, No. 5537, pp. 2037–2040.

Szalay and Gray 2004 — Szalay, A. and Gray, J. (2004). *Scientific Data Federation: The World-Wide Telescope*. In Foster, I. and Kesselman, C. (eds.), *The Grid 2*. San Francisco, CA: Morgan Kauffman.

Szyperski 2002 — Szyperski, C. (2002). *Component Software: Beyond Object-Oriented Programming*, 2nd edn. Reading, MA: Addison-Wesley.

Tanenbaum 2003 — Tanenbaum, A.S. (2003). *Computer Networks*, 4th edn. Upper Saddle Rivee, NJ: Prentice-Hall International.

Tanenbaum 2007 — Tanenbaum, A.S. (2007). *Modern Operating Systems*, 3rd edn. Englewood Cliffs, NJ: Prentice-Hall.

Tanenbaum and van Renesse 1985 — Tanenbaum, A. and van Renesse, R. (1985). Distributed operating systems. *Computing Surveys*, *ACM*, Vol. 17, No. 4, pp. 419–70.

Tanenbaum *et al*. 1990 — Tanenbaum, A.S., van Renesse, R., van Staveren, H., Sharp, G., Mullender, S., Jansen, J. and van Rossum, G. (1990). Experiences with the Amoeba distributed operating system. *Comms. ACM*, Vol. 33, No. 12, pp. 46–63.

Taufer *et al.* 2003 — Taufer, M., Crowley, M., Karanicolas, J., Cicotti, P., Chien, A. and Brooks, L. (2003). *Moving Towards Desktop Grid Solutions for Large Scale Modelling in Computational Chemistry*. University of California, San Diego.

Terry *et al.* 1995 — Terry, D., Theimer, M., Petersen, K., Demers, A., Spreitzer, M. and Hauser, C. (1995). Managing update conflicts in Bayou, a weakly connected replicated storage system. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles*, pp. 172–183.

TFCC — *IEEE Task Force on Cluster Computing*.

Thaler *et al.* 2000 — Thaler, D., Handley, M. and Estrin, D. (2000). *The Internet Multicast Address Allocation Architecture*. Internet RFC 2908

Thekkath *et al.* 1997 — Thekkath, C.A., Mann, T. and Lee, E.K. (1997). Frangipani: A scalable distributed file system, in *Proc. of the 16th ACM Symposium on Operating System Principles*, St. Malo, France, October, pp. 224–237.

Tokuda *et al*. 1990 — Tokuda, H., Nakajima, T. and Rao, P. (1990). Real-time Mach: Towards a predictable real-time system. In *Proceedings of the USENIX Mach Workshop*, pp. 73–82.

Topolcic 1990 — Topolcic, C. (ed.) (1990). *Experimental Internet Stream Protocol, Version* 2. Internet RFC 1190.

Tsoumakos and Roussopoulos 2006 — Tsoumakos, D. and Roussopoulos, N. (2006). Analysis and comparison of P2P search methods. In *Proceedings of the 1st international Conference on Scalable information Systems (InfoScale '06)*, Hong Kong, p.25.

Tzou and Anderson 1991 — Tzou, S.-Y. and Anderson, D. (1991). The performance of message-passing using restricted virtual memory remapping. *Software–Practice and Experience*, Vol. 21, pp. 251–67.

van Renesse *et al*. 1989 — van Renesse, R., van Staveran, H. and Tanenbaum, A. (1989). The performance of the Amoeba distributed operating system. *Software – Practice and Experience*, Vol. 19, No. 3, pp. 223–34.

van Renesse *et al.* 1995 — van Renesse, R., Birman, K., Friedman, R., Hayden, M. and Karr, D. (1995). A framework for protocol composition in Horus. In *Proceedings of the PODC 1995*, pp. 80–9.

van Renesse *et al.* 1996 — van Renesse, R., Birman, K. and Maffeis, S. (1996). Horus: A flexible group communication system. *Comms. ACM*, Vol. 39, No. 4, pp. 54–63.

van Renesse *et al.* 1998 — van Renesse, R., Birman, K., Hayden, M., Vaysburd, A. and Karr, D. (1998). Building adaptive systems using Ensemble. *Software–Practice and Experience*, Vol. 28, No. 9, pp. 963–979.

van Steen *et al.* 1998 — van Steen, M., Hauck, F., Homburg, P. and Tanenbaum, A. (1998). Locating objects in wide-area systems. *IEEE Communication*, Vol. 36, No. 1, pp. 104–109.

Vinoski 1998 — Vinoski, S. (1998). New features for CORBA 3.0. *Comms. ACM*, Vol. 41, No. 10, pp. 44–52.

Vinoski 2002 — Vinoski, S. (2002). Putting the 'Web' into Web Services. *IEEE Internet Computing*. Vol. 6, No. 4, pp. 90–92.

Vogels 2003 — Vogels, W. (2003). Web Services are not Distributed objects. *IEEE Internet Computing*, Vol. 7, No, 6, pp. 59–66.

Vogt *et al.* 1993 — Vogt, C., Herrtwich, R.G. and Nagarajan, R. (1993). HeiRAT – The Heidelberg Resource Administration Technique: Design Philosophy and Goals. *Kommunikation in verteilten Systemen*, Munich, Informatik aktuell, Springer.

Volpano and Smith 1999 — Volpano, D. and Smith, G. (1999). Language issues in mobile program security. In *Mobile Agents and Security*. Vol. 1419 in *Lecture Notes in Computer Science*, Springer-Verlag, pp. 25–43. .

von Eicken *et al.* 1995 — von Eicken, T., Basu, A., Buch, V. and Vogels, V. (1995). U-Net: A user-level network interface for parallel and distributed programming. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles*, pp. 40–53.

Wahl *et al.* 1997 — Wahl, M., Howes, T. and Kille, S. (1997). *The Lightweight Directory Access Protocol (v3)*. Internet RFC 2251.

Waldo 1999 — Waldo, J. (1999). The Jini architecture for network-centric computing. *Comms. ACM*, Vol. 42, No. 7, pp. 76–82.

Waldo *et al*. 1994 — Waldo, J., Wyant, G., Wollrath, A. and Kendall, S. (1994). A note on distributed computing. In Arnold *et al.* 1999, pp. 307–26.

Waldspurger *et al.* 1992 — Waldspurger, C., Hogg, T., Huberman, B., Kephart, J. and Stornetta, W. (1992). Spawn: A distributed computational economy. *IEEE Transactions on Software Engineering*, Vol. 18, No. 2, pp. 103–17.

Wang *et al*. 2001 — Wang, N., Schmidt, D. C. and O'Ryan, C. (2001). Overview of the CORBA component model. In Heineman, G. T. and Councill, W.T. (eds), *Component-Based Software Engineering: Putting the Pieces Together*, Addison-Wesley Longman Publishing Co., Boston, MA: Addison-Wesley, pp. 557–571.

Want 2004 — Want, R. (2004). Enabling ubiquitous sensing with RFID. *IEEE Computer*, Vol. 37, No. 4, pp. 84–86.

Want and Pering 2003 — Want, R. and Pering, T. (2003). New horizons for mobile computing. In *Proceedings of the First IEEE International Conference on Pervasive Computing and Communication (PerCom'03)*, Dallas-Fort Worth, TX, March, pp. 3–8.

Want *et al*. 1992 — Want, R., Hopper, A., Falcao, V. and Gibbons, V. (1992). The Active Badge location system. *ACM Transactions on Information Systems*, Vol. 10, No.1, pp. 91–102.

Want *et al*. 2002 — Want, R., Pering, T., Danneels, G., Kumar, M., Sundar, M. and Light, J. (2002). The personal server: Changing the way we think about ubiquitous computing. In *Proceedings of the Fourth International Conference on Ubiquitous Computing (Ubicomp 2002)*, Goteborg, Sweden, Sep.–Oct., pp.194–209.

Weatherspoon and Kubiatowicz 2002 — Weatherspoon, H. and Kubiatowicz, J.D. (2002). Erasure coding vs. replication: A quantitative comparison. *1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, Cambridge, MA, March, pp. 328–38.

web.mit.edu I — *Kerberos: The Network Authentication Protocol*.

web.mit.edu II — *The Three Myths of Firewalls*.

Wegner 1987 — Wegner, P. (1987). Dimensions of object-based language design. *SIGPLAN Notices*, Vol. 22, No. 12, pp. 168–182.

Weikum 1991 — Weikum, G. (1991). Principles and realization strategies of multilevel transaction management. *ACM Transactions on Database Systems*, Vol. 16, No. 1, pp. 132–40.

Weiser 1991 — Weiser, M. (1991). The computer for the 21st Century. *Scientific American*, Vol. 265, No. 3, pp. 94–104.

Weiser 1993 — Weiser, M. (1993). Some computer science issues in ubiquitous computing. *Comms. ACM*, Vol. 36, No. 7, pp. 74–84.

Wellner 1991 — Wellner, P.D. (1991). The DigitalDesk calculator – tangible manipulation on a desk-top display. In *Proceedings of the 4th Annual ACM Symposium on User Interface Software and Technology*, Hilton Head, SC, November, pp. 27–33.

Wheeler and Needham 1994 — Wheeler, D.J. and Needham, R.M. (1994). TEA, a Tiny Encryption Algorithm. Technical Report 355, *Two Cryptographic Notes*, Computer Laboratory, University of Cambridge, December, pp. 1–3.

Wheeler and Needham 1997 — Wheeler, D.J. and Needham, R.M. (1997). *Tea Extensions*. October 1994, pp. 1–3.

Whitaker *et al*. 2002 — Whitaker, A., Shaw, M. and Gribble, D.G. (2002). Denali: Lightweight virtual machines for distributed and networked applications. *Technical Report 02-02-01*, University of Washington.

Wiesmann *et al.* 2000 — Wiesmann, M., Pedone, F., Schiper, A., Kemme, B. and Alonso, G. (2000). Understanding replication in databases and distributed systems. In *Proceedings of the 20th International Conference on Distributed Computing Systems (ICDCS '2000)*, Taipei, Republic of China, p. 464.

Williams 1998 — Williams, P. (1998). JetSend: An appliance communication protocol. In *Proceedings of the IEEE International Workshop on Networked Appliances, (IEEE IWNA '98)*, Kyoto, Japan, November, pp. 51–53.

Winer 1999 — Winer, D. (1999). *The XML-RPC specification.*

Wobber *et al.* 1994 — Wobber, E., Abadi, M., Burrows, M. and Lampson, B. (1994). Authentication in the Taos operating system. *ACM Transactions on Computer Systems.* Vol. 12, No. 1, pp. 3–32.

Wright *et al*. 2002 — Wright, M., Adler, M., Levine, B.N. and Shields, C. (2002). An analysis of the degradation of anonymous protocols. In *Proceedings of the Network and Distributed Security Symposium (NDSS '02)*, February.

wsdl4j.sourceforge.org — *The Web Services Description Language for Java Toolkit (WSDL4J*).

Wulf *et al.* 1974 — Wulf, W., Cohen, E., Corwin, W., Jones, A., Levin, R., Pierson, C. and Pollack, F. (1974). HYDRA: The kernel of a multiprocessor operating system. *Comms. ACM*, Vol. 17, No. 6, pp. 337–345.

Wuu and Bernstein 1984 — Wuu, G.T. and Bernstein, A.J. (1984). Efficient solutions to the replicated log and dictionary problems. In *Proceedings of the Third Annual Symposium on Principles of Distributed Computing*, pp. 233–42.

www.accessgrid.org — *The Access Grid Project.*

www.adventiq.com — Adventiq Ltd. *Home page* featuring their KVM-over-IP technology.

www.akamai.com — Akamai. *Home page*.

www.apple.com I — Apple Computer. *Bonjour Protocol Specifications*.

www.apple.com II — Apple Computer. iChat v*ideo conferencing for the rest of us*.

www.beowulf.org — The Beowulf Project. *Resource centre*.

www.bittorrent.com — The Official BitTorrent Website.

www.bluetooth.com — The Official Bluetooth SIG Website.

www.butterfly.net — *The scalable, reliable and high-performance online game platform, GoGrid*.

www.bxa.doc.gov — Bureau of Export Administration, US Department of Commerce. *Commercial Encryption Export Controls*.

www.cdk5.net — Coulouris, G., Dollimore, J. and Kindberg, T. (eds.). *Distributed Systems, Concepts and Design: Supporting material*.

www.citrix.com — Citrix Corporation. *Citrix XenApp.*

www.conviva.org — Conviva. *Home page*.

www.coralcdn.org — Coral Content Distribution Network. *Home page*.

www.cren.net — Corporation for Research and Educational Networking. *CREN Certificate Authority*.

www.cryptopp.com — Crypto++® Library 5.2.1.

www.cs.cornell.edu — The 3rd ACM SIGOPS International Workshop on Large Scale Distributed Sytems and Middlewqare (LADIS'09). *Keynote by Jeff Dean on Large-Scale Distributed Systems at Google: Current Systems and Future Directions*.

www.cs.york.ac.uk/dame — *Distributed Aircraft Maintenance Environment (DAME)*.

www.cuseeme.com — CU-SeeMe Networks Inc. *Home page*.

www.dancres.org — Blitz open source project. *Home page*.

www.doi.org — International DOI Foundation. *Pages on digital object identifiers*.

www.dropbox.com — Dropbox file hosting service. *Home page*.

www.dtnrg.org — Delay Tolerant Networking Research Group. *Home page*.

www.gigaspaces.com — GigaSpaces. *Home page*.

www.globalcrossing.net — Global Crossing. *IP Network Performance – monthly history*.

www.globexplorer.com — *Globexplorer, the world's largest online library of aerial and satellite imagery*.

www.globus.org — The Globus Project. *Latest stable version of the Globus Toolkit*.

www.google.com I — Google. *Google Apps*.

www.google.com II — Google. *Google Maps*.

www.google.com III — Google. *Google Corporate Information*.

www.google.com IV — Google. *Google Corporate Information. Design Principles.*

www.gridmpi.org — The GridMPI Project. *Home page*.

www.handle.net — Handle system. *Home page*.

www.iana.org I — Internet Assigned Numbers Authority. *Home page*.

www.iana.org II — Internet Assigned Numbers Authority. *IPv4 Multicast Address Space Registry.*

www.ibm.com — IBM. *WebSphere Application Server home page*.

www.ietf.org — Internet Engineering Task Force. *Internet RFC Index page*.

www.iona.com — Iona Technologies. *Orbix*.

www.ipnsig.org — InterPlaNetary Internet Project. *Home page.*

www.ipoque.com — Ipoque GmbH. *Internet Study 2008/2009.*

www.isoc.org — Robert Hobbes Zakon. *Hobbe's Internet Timeline*.

www.jbidwatcher.com — JBidwatcher Project. *Home page.*

www.jboss.org — JBoss Open Source Community. *Home page.*

www.jgroups.org — JGroups Project. *Home page.*

www.json.org — JSON external data representation. *Home page*.

www.kontiki.com — Kontiki Delivery Management System. *Home page*.

www.microsoft.com I — Microsoft Corporation. *Active Directory Services*.

www.microsoft.com II — Microsoft Corporation. *Windows 2000 Kerberos Authentication*, White Paper.

www.microsoft.com III — Microsoft Corporation. *NetMeeting home page*.

www.microsoft.com IV — Microsoft Corporation. *Azure home page*.

www.mozilla.org — Netscape Corporation. *SSL 3.0 Specification*.

www.mpi-forum.org — Message Passing Interface (MPI) Forum. *Home page*.

www.neesgrid.org — *NEES Grid, Building the National Virtual Collaboratory for Earthquake Engineering*.

www.netscape.com — *Netscape*. Home page.

www.nfc-forum.org — Near Field Communication (NFC). *Forum home page*.

www.oasis.org — Web Services reliable messaging. *WS-Reliablemessaging. Vn 1.1. Oasis standard.*

www.omg.org — Object Management Group. *Index to CORBA services.*

www.opengroup.org — Open Group. *Portal to the World of DCE.*

www.openmobilealliance.org — Open Mobile Alliance. *Home page*.

www.openssl.org — OpenSSL Project. *OpenSSL: The Open Source toolkit for SSL/TLS.*

www.openstreetmap.org — OpenStreetMap. *Home page*.

www.osgi.org — The OSGi Alliance. *Home page*.

www.ow2.org — The OW2 Consortium. *Home page*.

www.parc.com — PARC Forum. *Presentation to Marissa Mayer, VP Google.*

www.pgp.com PGP. *Home page.*

www.progress.com Apama from Progress Software. *Home page.*

www.prototypejs.org Prototype JavaScript Framework. *Home page*.

www.realvnc.com RealVNC Ltd. *Home page*.

www.redbooks.ibm.com IBM Redbooks. *WebSphere MQ V6 Fundamentals*.

www.reed.com Read, D.P. (2000). *The End of the End-to-End Argument*.

www.research.ibm.com Gryphon Project. *Home page*.

www.rsasecurity.com I RSA Security Inc. *Home page*.

www.rsasecurity.com II RSA Corporation (1997). *DES Challenge*.

www.rsasecurity.com III RSA Corporation (2004). *RSA Factoring Challenge*.

www.rtj.org *Real-Time for Java TM Experts Group.*

www.secinf.net Network Security Library.

www.sei.cmu.edu Software Engineering Instute, Carnegie Mellon. *Home page of the Ultra Large Systems (ULS) Initiative.*

www.smart-its.org The Smart-Its Project. *Home page*.

www.spec.org *SPEC SFS97 Benchmark*.

www.springsource.org SpringSource Community. *Spring Framework*.

www.upnp.org Universal Plug and Play. *Home page*.

www.us.cdnetworks.com CDNetworks Inc. *Home page*.

www.uscms.org USCMS, *The Compact Muon Solenoid*.

www.verisign.com Verisign Inc. *Home page*.

www.w3.org I World Wide Web Consortium. *Home page*.

www.w3.org II World Wide Web Consortium. *Pages on the HyperText Markup Language*.

www.w3.org III World Wide Web Consortium. *Pages on Naming and Addressing*.

www.w3.org IV World Wide Web Consortium. *Pages on the HyperText Transfer Protocol*.

www.w3.org V World Wide Web Consortium. *Pages on the Resource Description Framework and other metadata schemes*.

www.w3.org VI World Wide Web Consortium. *Pages on the Extensible Markup Language*.

www.w3.org VII World Wide Web Consortium. *Pages on the Extensible Stylesheet Language*.

www.w3.org VIII *XML Schemas*. W3C Recommendation (2001).

www.w3.org IX World Wide Web Consortium. *Pages on SOAP.*

www.w3.org X World Wide Web Consortium. *Pages on Canonical XML, Version 1.0*. W3C Recommendation.

www.w3.org XI — World Wide Web Consortium. *Pages on Web Services Description Language (WSDL)*.

www.w3.org XII — World Wide Web Consortium. *Pages on XML Signature Syntax and Processing*.

www.w3.org XIII — World Wide Web Consortium. *Pages on XML key management specification (XKMS)*.

www.w3.org XIV — World Wide Web Consortium. *Pages on XML Encryption Syntax and Processing*.

www.w3.org XV — World Wide Web Consortium. *Pages on Web Services Choreography Requirements*. W3C Working Draft.

www.w3.org XVI — Burdett, D. and Kavantsas, N. *WS Choreography Model Overview*. W3C Working Draft.

www.w3.org XVII — World Wide Web Consortium. *Pages on Web Services Choreography Description Language Version 1.0*.

www.w3.org XVIII — World Wide Web Consortium. *Pages on Web Services Choreography Interface (WSCI)*.

www.w3.org XIX — World Wide Web Consortium. *Pages on Device Independence*.

www.w3.org XX — World Wide Web Consortium. *Pages on the Semantic Web*.

www.w3.org XXI — World Wide Web Consortium. *Pages on the XML Binary Charaterization Working Group*.

www.w3.org XXII — World Wide Web Consortium. *Pages on the SOAP Message Transmission Optimization Protocol recommendations*.

www.w3.org XXIII — World Wide Web Consortium. *Pages on the WS-Addressing Working Group*.

www.w3.org XXIV — World Wide Web Consortium. *Pages on the Geolocation API Specification*.

www.wapforum.org — WAP Forum. *White Papers and Specifications*.

www.wlana.com — *The IEEE 802.11 Wireless LAN Standard*.

www.xbow.com — Crossbow Technology Inc. *Pages on wireless sensor networks*.

www.xen.org — Xen open source community. *Home page*.

www.zeroconf.org — IETF Zeroconf Working Group. *Home page*.

Wyckoff *et al.* 1998 — Wyckoff, P., McLaughry, S., Lehman, T. and Ford, D. (1998). T Spaces. *IBM Systems Journal*, Vol. 37, No. 3.

Xu and Liskov 1989 — Xu, A. and Liskov, B. (1989). The design for a fault-tolerant, distributed implementation of Linda. In *Proceedings of the 19th International Symposium on Fault-Tolerant Computing*, Chicago, IL, June, pp. 199–206.

zakon.org — Zakon, R.H. Hobbes' Internet Timeline v7.0,

Zhang and Kindberg 2002 — Zhang, K. and Kindberg, T. (2002). An authorization infrastructure for nomadic computing. In *Proceedings of the Seventh ACM Symposium on Access Control Models and Technologies*, Monterey, CA, June, pp. 107–113.

Zhang *et al.* 1993 — Zhang, L., Deering, S.E., Estrin, D., Shenker, S. and Zappala, D. (1993). RSVP – A new resource reservation protocol. *IEEE Network Magazine*, Vol. 9, No. 5, pp. 8–18.

Zhang *et al.* 2005a — Zhang, H., Goel, A., and Govindan, R. (2005). Improving lookup latency in distributed hash table systems using random sampling. *IEEE/ACM Trans. Netw*. 13, 5 (Oct. 2005), 1121–1134.

Zhang *et al.* 2005b — Zhang, X., Liu, J., Li, B. and Yum, T.-S. (2005). CoolStreaming/DONet: A data-driven overlay network for live media streaming. In *Proceedings of IEEE INFOCOM'05*, Miami, FL, USA, March, pp. 2102–2011.

Zhao *et al.* 2004 — Zhao, B.Y., Huang, L., Stribling, J., Rhea, S.C., Joseph, A.D. and Kubiatowicz, J.D. (2004). Tapestry: A resilient global-scale overlay for service deployment, *IEEE Journal on Selected Areas in Communications*, Vol. 22, No. 1, pp. 41–53.

Zimmermann 1995 — Zimmermann, P.R. (1995). *The Official PGP User's Guide*. MIT Press.